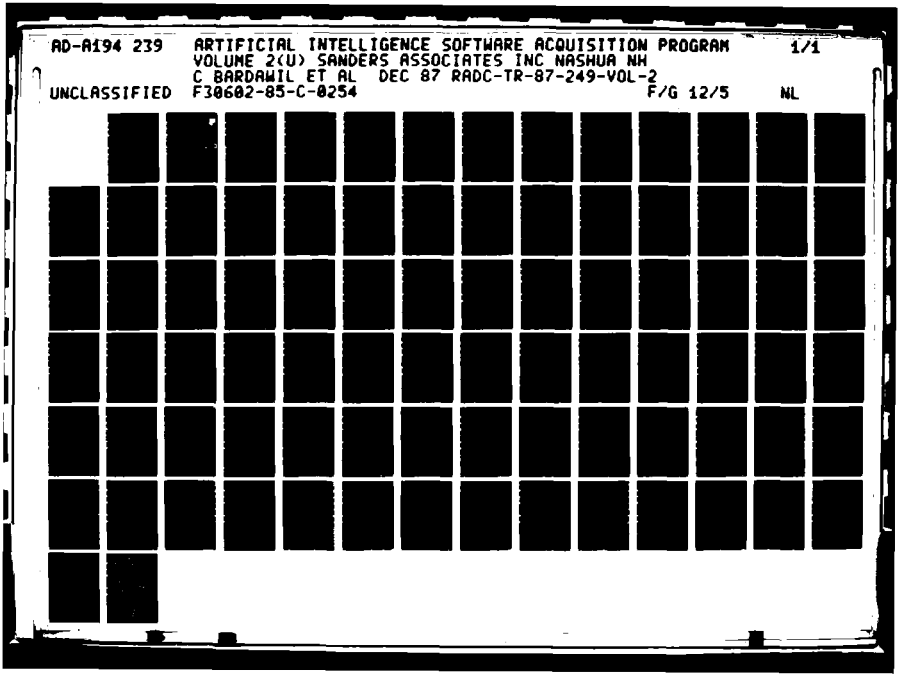
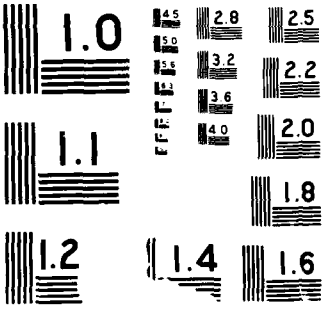

AD-A194 239
ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM
1/1
VOLUME 2(U) SANDERS ASSOCIATES INC NASHUA NH
C BARDAWIL ET AL DEC 87 RADC-TR-87-249-VOL-2
UNCLASSIFIED
F30602-85-C-0254
F/G 12/5
NL


1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

RADC-TR-87-249, Vol II (of two)
Final Technical Report
December 1987

ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM

Sanders Associates, Inc.

Carol Bardawil, Larry Fry, Sandy King, Linda Leszcynski and Graham O'Neil

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-87-249, Vol II (of two) has been reviewed and is approved for publication.

APPROVED:

RICHARD M. EVANS
Project Engineer

APPROVED:

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

A/ - 239

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS: N/A
2a SECURITY CLASSIFICATION AUTHORITY: N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): N/A	5 MONITORING ORGANIZATION REPORT NUMBER(S): RADC-TR-87-249, Vol II (of two)
6a NAME OF PERFORMING ORGANIZATION: Sanders Associates, Inc. / 6b OFFICE SYMBOL (If applicable):	7a NAME OF MONITORING ORGANIZATION: Rome Air Development Center (COEE)
6c ADDRESS (City, State, and ZIP Code): 95 Canal Street, CS 2004, Nashua NH 03061-2004	7b ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700
8a NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center / 8b OFFICE SYMBOL (If applicable): COEE	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F30602-85-C-0254
8c ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO: 63728F; PROJECT NO: 2532; TASK NO: 01; WORK UNIT ACCESSION NO: 16

11 TITLE (Include Security Classification)
ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM

12 PERSONAL AUTHOR(S)
Carol Bardawil, Larry Fry, Sandy King, Linda Leszcynski, Graham O'Neil

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Aug 85 TO Aug 87	December 1987	92

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Software development process, software acquisition model, knowledge-based systems, documentation standards.
12	05		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The goal of this research was to evaluate the software development process for artificial intelligence (AI) systems and postulate a software acquisition model. To accomplish this research, the major elements performed were a literature search, a case study analysis of 26 knowledge based system (KBS) development efforts, and consultation with experienced AI system developers. The results of this study are presented in a two volume report. Volume I presents observations made during the analysis of KBS software developments and provides summaries of the case study data. A comparison of the KBS development process to DOD-STD-2167 is also documented. Volume II discusses a KBS process model and customer/developer interface model. A comparison of the postulated model with DOD-STD-2167 and DOD-STD-2167A (draft) is made in terms of activities, products, reviews and baselines.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: RICHARD M. EVANS	22b TELEPHONE (Include Area Code): (315) 330-3564 / 22c OFFICE SYMBOL: RADC (COEE)

DD Form 1473, JUN 86 — Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Contents

Volume II

List of Figures

Volume II

List of Tables

Volume II

List of Contributers

The following people have contributed to the Artificial Intelligence Software Acquisition Program (AISAP) study and final report.

Case Study Participants

Marilyn Aglubat	Northrop Avionics Division
Sam Ashby	Boeing Military Airplane Company
Virginia Barker	Digital Equipment Corporation
Carlos Bhola	Expert Technologies, Inc.
Dr. R. P. Bonasso	The MITRE Corporation (McLean, VA)
Rodney M. Bond	ARINC Research Corporation
Douglas Clafin	Lockheed Aircraft Service Company
Stan Coffman	Lockheed-Georgia Company
P. R. Deweese	Lockheed-Georgia Company
Linda Dudding	Lockheed Aircraft Service Company
Vicki Florian	Software Architecture and Engineering, Inc.
Kimberly Freitas	Inference Corporation
Kermit Gates	PAR Government Systems Corporation
Terry Ginn	Sanders Associates, Inc.
Gary G. Greenfield	Frey Associates, Inc.
Carl Gunther	Inference Corporation
William B. Harrelson	Brattle Research Corporation
David Harris	Sanders Associates, Inc.
Dr. D. F. Hubbard	Carnegie Group, Inc.
Ted Jardine	Boeing Computer Services
Elizabeth Kooker	IBM Federal Systems Group
Robert Lough	Northrop Avionics Division
Jim Montague	Texas Instruments, Inc.
Edward Orciuch	Digital Equipment Corporation
Laurent Piketty	Inference Corporation
M. J. Prelle	The MITRE Corporation (Bedford, MA)
Jack Rahaim	Digital Equipment Corporation
Ethan Scarl	The MITRE Corporation (Bedford, MA)
Anthony D. Vanker	GTE Data Services

Reviewers/Consultants

Dick Cloutier	Sanders Associates, Inc.
Terry Ginn	Sanders Associates, Inc.
David Harris	Sanders Associates, Inc.
Dr. Charles Rich	Massachusetts Institute of Technology
Tom Royer	Sanders Associates, Inc.

Preface

In August 1985, the Rome Air Development Center selected Sanders Associates, Inc. to evaluate the software development process for Artificial Intelligence (AI) systems and postulate a software acquisition model. To accomplish these objectives, Sanders devised a strategy consisting of the following major elements:

- Literature review;
- Case study analysis; and
- Consultation with experienced AI system developers.

The case study analyses represent historical data on 26 knowledge base system (KBS) development efforts. Because the case data focuses on KBS software, the acquisition model developed pertains to KBS efforts.

The results of this study are presented in a two volume report. Volume I presents observations made during the analysis of KBS software developments as well as summaries of the case study data. A comparison of the KBS development process to DOD-STD-2167 is also made.

Volume II presents a KBS process model as well as a postulated customer/developer interface model. A comparison of the postulated model with DOD-STD-2167 and DOD-STD-2167A (draft) is made in terms of activities, products, reviews and baselines.

SECTION 1

Conventional Software Development Methodology

1.1 Description of DOD-STD-2167

Large scale Department of Defense (DOD) computerized system developments, which began in earnest in the 1960's, have taken many approaches to the generation of software. Although there were no formally defined models for software development within DOD, an acceptable approach evolved that became the pattern for software developments acquired under contract to the DOD. This approach documented by [3, Boehm], [28, Metzger], and others was characterized as the waterfall software development model. Although the terminology and the break points between phases varied, the models essentially included the phases of:

- requirements definition;
- design;
- coding and debug;
- integration and testing; and
- operations and maintenance.

During the 1970's, a number of activities took place to adopt the waterfall development model as the methodical way to produce software within DOD. The Air Forces 800-14 Regulation and the Navy Military Standard 1679 are two examples of the service's desire to manage the software development process against the backdrop of the waterfall model.

By the late 1970's, numerous development approaches had evolved and it appeared that many of the DOD agencies were intent on specifying their own unique requirements for developing software. As a result, more and more approaches evolved and industry was faced with managing software developments using management control systems which were unique to each customer.

In an attempt to bring the Services together, the Joint Logistics Commanders (JLC) Computer Resource Management Group conducted a workshop in April 1979 to address the management and control issues concerning DOD software developments. The primary output of the workshop was the proposition that the Services could and should be developing software under a single set of policies and procedures and that the commonalities among software applications far exceeded any unique requirements. The workshop recognized the waterfall model as a proper starting point and recommended that policy and standards be developed which defined the software development process. This recommendation prompted the DOD to embark on a standards effort that yielded DOD-STD-2167, 24 data item descriptions (DIDs), and changes to MIL-STD-483, 490 and 1521 on 5 June 1985. A number of open issues remained when DOD-STD-2167 was published. These issues

have been addressed in a Draft Revision A to the basic standard which was released by DOD for review on 1 April 1987.

The software development model under DOD-STD-2167 was patterned after the waterfall model and included the concepts of activities, products, reviews and baselines. The model represents software development from the government viewpoint, and as such, visualizes the development process as a series of sequential phases with reviews and documentation integral to each phase of development.

The 2167 waterfall model was developed to provide visibility into the software development process and control mechanisms over the evolving product as development occurred. Heavy emphasis was placed on configuration identification, through the concept of computer software configuration items (CSCIs), and configuration control. The 2167 requirements for baseline establishment represents a control mechanism for requirements, design, and code as it evolves. Associated with the establishment of baselines are various reviews aimed at assessing software development progress at various phase points (e.g., requirements analysis, preliminary design) and determining readiness for baseline control.

Another aspect of the 2167 model was the need for flexibility in application and compatibility with other disciplines. The goal was to develop a standard that could be *tailored* to various application needs. Through such an approach, 2167 would provide the framework within which software development could occur but at the same time be applicable to the diverse world of DOD applications. In addition to being flexible, the standard should tie to standards in other disciplinary areas of development to include configuration management and system engineering. The changes to Military Standards 483, 490 and 1521 were made for compatibility with the DOD-STD-2167 model for software development.

1.1.1 Disciplined Software Development

DOD-STD-2167 advocates a disciplined approach to software development. Disciplined development involves breaking down the system into manageable pieces and phases that allow for progressive control and visibility into project cost and schedule, and applying the structured engineering principles advocated in the 1970's. Breaking down the system allows multiple developers to work in parallel, which is necessary for large projects. The process generally involves a Definition and Analysis, Design, Implementation and Testing Phase and includes structured inplementation and top-down concepts.

The Definition Phase requires that the software developer define *what* capabilities the software should provide, and structure these requirements so they are clear, correct and traceable. The information established during requirements definition is documented in a Software Requirements Specification which serves as a foundation or baseline for the entire system. Namely, the software is developed with the intent of specifically implementing the defined requirements.

The Software Design Phase examines the defined requirements and provides a solution on *how* the system should be developed. Major aspects of the Design Phase are top-down software design, interface management control, and software test planning. The main goal of top-down software design resides in the stepwise refinement of the solutions into functional and logical components. In the design, one starts with the most inclusive set of functions and decomposes each until all functions

have been accounted for systematically. Results of top-down software design are represented in a Design Specification which contains a functional description of *what* the system does and a logical description of *how* the system is structured. The Design Specification serves as a baseline from which future detailed design and coding stem. Structuring tools such as Hierarchical Input Processing Output (HIPO) or Program Design Languages (PDL) are encouraged to aid in the documentation process. Another aspect of the Design Phase is the definition and documentation of interfaces between components. A precise definition of the interface supports and complements the modularization of the system. Software testing is also defined during the Design Phase and includes test plans for module integration, system acceptance and site tests for the system.

Implementation relies heavily on top-down techniques and the Structured Programming *philosophy* of writing programs according to a set of rigid rules in order to decrease testing problems, increase productivity, and increase the readability of the resulting program. [28, Metzger]. The thrust of structured programming revolves around the need for clarity, order and readibility to allow for error-free code that can be understood by any properly trained individual. To encourage readable code, certain coding conventions are adopted that eliminate most unconditional branches and allow programs to be developed in a logical sequence, with supporting comments.

The Testing Phase, although often overlooked, is as critical as any other phase because the consequences of ineffective testing can result in a defective product and/or a dissatisified customer. The main goal of this phase involves thorough testing of the system's capabilities against requirements. Oftentimes, customer training is also incorporated into this phase.

1.1.2 Activities, Products

The following subsections present a brief discussion of the activities which are part of the five phases of a system's life cycle according to DOD-STD-2167:

- Software Requirements;
- Design;
- Coding and Unit Testing;
- Computer Software Component (CSC) Integration and Testing; and
- CSCI Testing.

1.1.2.1 Software Requirements Analysis

There are a number of activities performed during the requirements analysis phase of software development. These activities include the definition of software requirements, development planning, and documentation preparation and updating.

The primary thrust in requirements definition is to specify the functional, performance, interface and qualification requirements for a manageable entity of software which is commonly referred to

as a Computer Software Configuration Item (CSCI). Included as part of this activity is the definition of design and programming constraints, quality requirements and requirements concerning deliverables.

Development planning involves the identification of resources, organization, schedules and standards to be applied to the software development process. This planning encompasses the project management, engineering, configuration management and quality assurance disciplines. The plan also specifies the extent of involvement in each discipline. Heavy emphasis is placed on the control aspects of the program in terms of product configuration control for deliverable as well as non-deliverable software, interface control between system components, and control over project teams to include subcontractor arrangements.

1.1.2.2 Design

A variety of activities are associated with the Design Phase which includes a definition of the design, modularization of the CSCIs, interface definition, configuration management and preparation of test requirements.

The Design Phase is generally divided into two parts: a preliminary phase and a detailed phase. Design is characterized by the development of an approach that includes mathematical models, functional flows and detailed flows to represent and keep track of the breakdown of CSCIs into system components. Software requirements are progressively refined and allocated to lower level components. The process within components is defined and an interface relationship between components is established. Once documentation is complete, a system's high level design is evaluated against the requirements and the low level design is evaluated against the high level design.

During the Design Phase, test requirements, responsibilities and schedules for informal testing should be identified and documented in a Software Development Folder (SDF) for configuration management purposes. The information tracked by the SDF includes requirements, design considerations and constraints, schedule, status information and test documentation. Test plans should be developed by the contractor for both formal and informal testing. Resources required for testing should also be determined.

1.1.2.3 Coding and Unit Testing

Some of the major goals associated with the Coding and Unit Testing Phase are coding and testing of components within the CSCI, configuration management, storage of test results, and preparation of test plans.

Components are tested in top-down sequence unless other methodologies have been approved by the contracting agency. All test results are recorded in the SDF. If changes are required as a result of the testing exercise, both the design and code should be recorded and updated in the SDF.

Detailed test procedures to conduct informal Computer Software Component (CSC) integration tests should be developed. Also, test procedures and a means to analyze test results should be provided for each CSCI.

1.1.2.4 Computer Software Component (CSC) Integration and Testing

The major goal of Computer Software Component (CSC) Integration and Testing is to integrate and test aggregates of coded components. The purpose of the phase revolves around the philosphy that components should be tested according to documented integration test plans, test descriptions and test procedures. Likewise, a review of test results, and test plans, descriptions, and procedures to fully test implemented software should be maintained and documented. Any software changes based on testing should be recorded in the design documentation and code as well as updated in the SDF.

Another aspect of CSC Integration and Testing involves computer memory and processing speed. As components are integrated, memory and processing time values should be compared with allocations determined during the Design Phase.

1.1.2.5 CSCI Testing

The main activities of the CSCI Testing Phase involve testing the fully implemented CSCI. Essentially, the objective is to prove that requirements are adequately and appropriately met by the software. Formal tests should be performed on each CSCI in accordance with the formal test cases already documented. Results of formal CSCI tests should be recorded in both summary and detail form. A detailed test history, evaluation of test results, test procedure deviations and recommendations should also be documented. Design and Coding changes based on component test results should be recorded in the design documentation and code and updated in the SDF.

1.1.3 Reviews, Baselines

DOD-STD-2167 identifies standards by which deliverable and non-deliverable software can be identified, specified and managed as configuration items throughout a system's life cycle. The standard provides for government control and awareness over software configuration items by having the contractor establish three main baselines: Functional, Allocated and Product. A baseline is comprised of a configuration identification document or a set of documents that are produced at planned intervals during a configuration item's life cycle. Completion of the Functional Baseline occurs at the end of the Pre Software Development Phase; the Allocated Baseline is the product of the Requirements Analysis Phase and the Product Baseline marks the end of the Testing Phase. The military standard also calls for the contractor to maintain internal control over deliverable and non-deliverable documents and code throughout the software development process by applying configuration management at the Preliminary Design, Detailed Design, Coding and Unit Testing, Computer Software Component (CSC) Integration and Testing, and the CSCI Testing Phase.

The next subsections address 2167's view on

- baselines;
- configuration management;
- reviews; and

- documents produced within a system's life cycle.

Figure 1-1/ 1-2 was extracted from the 5 June 1985 DOD-STD-2167 and depicts the reviews, baselines, configuration management and documents that are part of the system development process. Many of the documents will not be referenced in the following sections, but are requirements of the standard.

1.1.3.1 Pre Software Development

The Functional Baseline which contains the System Segment Specification (SSS) is established no later than the System Design Review (SDR). The SDR consists of a review to assess the optimization, correlation, completeness, and risks associated with the allocated technical requirements. Another aspect of the review is to determine whether the system engineering process that produced the allocated technical requirements of the engineering plan for the next phase is appropriate. The review continues until a definition of system characteristics occurs and configuration items are identified.

1.1.3.2 Software Requirements Analysis

The Allocated Baseline for each CSCI is established upon completion of the Software Specification Review (SSR) and upon authentication by the contracting agency. The purpose of the SSR is to review the Operational Concept Document (OCD) for the system, the Software Requirements Specification (SRS) and the Interface Requirements Specification (IRS) for each CSCI. The success of an SSR is based upon an approval of the SRS, IRS and OCD as forming an adequate foundation for continuing into the preliminary software design. In many cases, the IRS is not prepared as a separate document; instead the interface requirements are included in the SRS.

1.1.3.3 Preliminary Design

During the Preliminary Design Review (PDR), the contractor presents the Software Top Level Design Document (STLDD) and the Software Test Plan (STP) for each CSCI together with preliminary versions of the Computer System Operator's Manual (CSOM), the Software User's Manual (SUM), the Computer System Diagnostic Manual (CSDM), and the Computer Resources Integrated Support Document (CRISD) for review by the contracting agency. The PDR's purpose is to review the top-level design, test plans and preliminary operation and support documents with the contracting agency. Upon successful completion of the PDR, a contractor establishes the Developmental Configuration with the STLDD representing the preliminary design baseline within the Developmental Configuration.

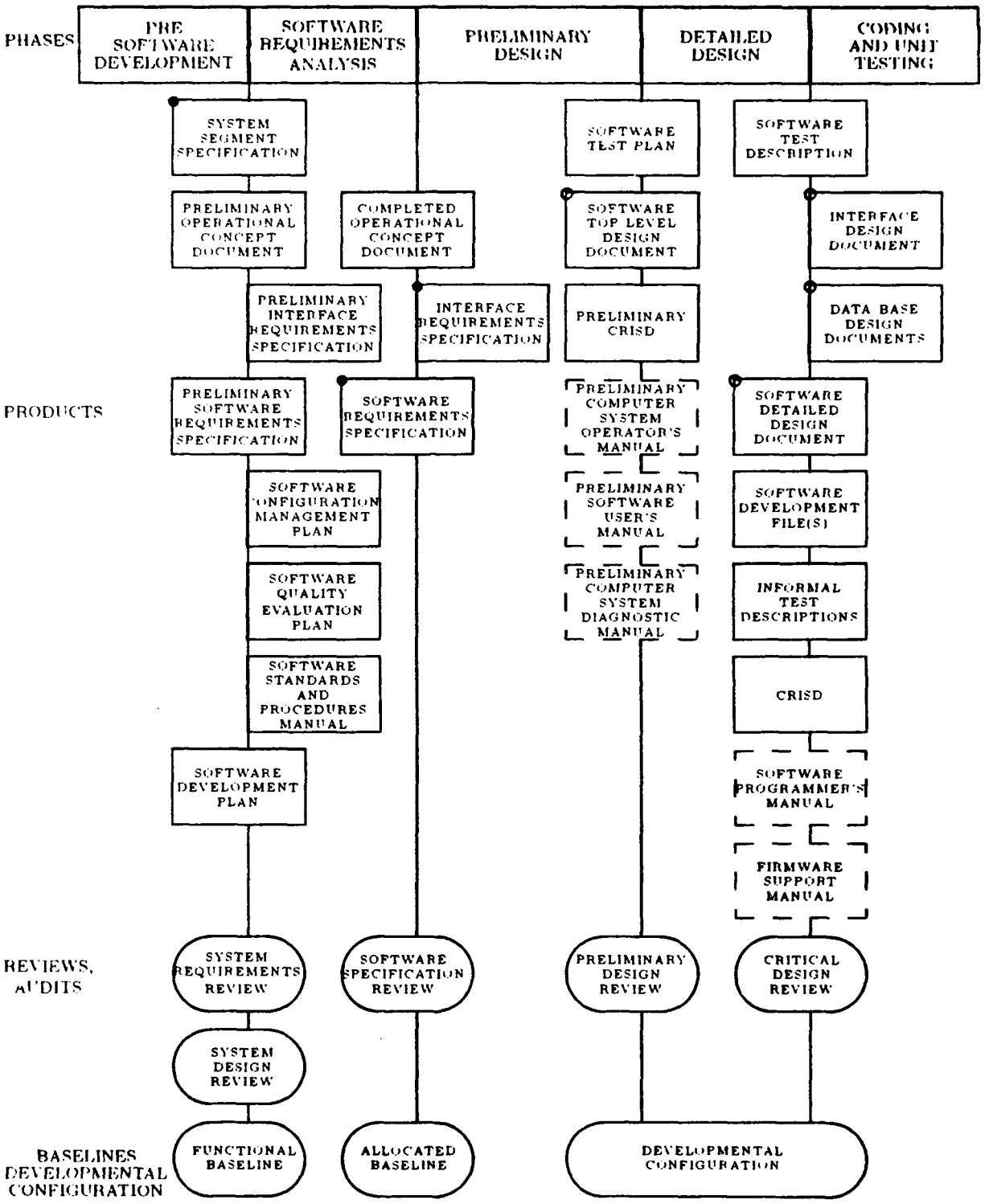


Figure 1-1: Software Development Cycle (per DOD-STD-2167)

1.1.3 Reviews, Baselines

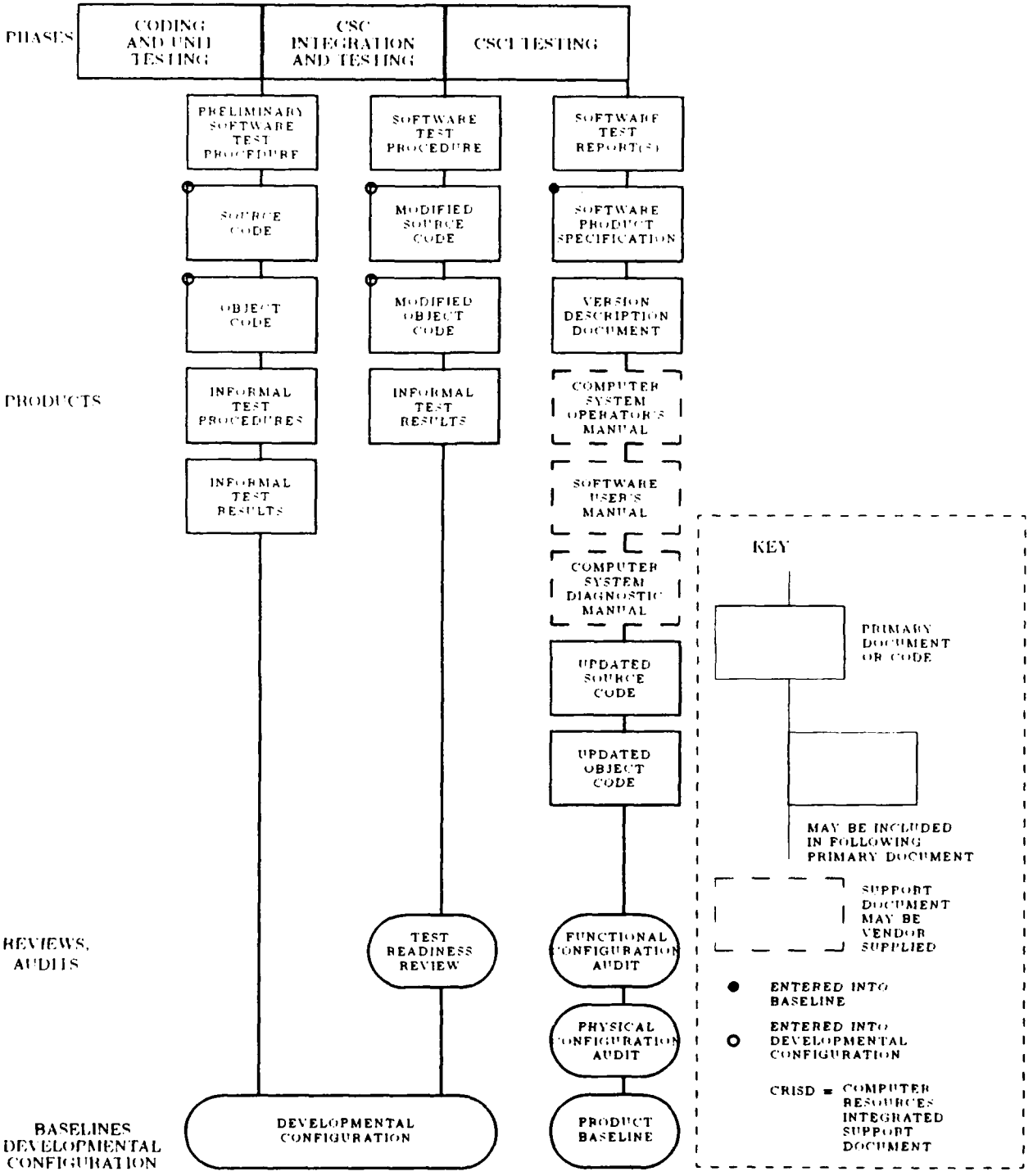


Figure 1-2: Software Development Cycle (Cont.) (per DOD-STD-2167)

1.1.3.4 Detailed Design

The Critical Design Review (CDR) is established to review the Software Detailed Design Document (SDDD) and the Software Test Description (STD) document for each CSCI. Other documents presented by the contractor are the Interface Design Documents (IDDs), the Data Base Design Documents (DBDDs), the Software Programmer's Manual (SPM), the Firmware Support Manual (FSM), the updated CSOM, the SUM, the CSDM and the final CRISD. The main goal of the CDR is to examine the detailed design, test description, and operational and support documents with the contracting agency. Once the CDR is successfully complete, the SDDD is entered into the Developmental Configuration for each CSCI.

1.1.3.5 Coding and Unit Testing

After each component is successfully tested and reviewed, the contractor can enter updated design documentation, source and object code, and associated listings for the component into the Developmental Configuration for the CSCI. Other than internal reviews, no major review governs the manner in which source and object code is added to the Developmental Configuration. Reviews are internal in nature and based on evolving source code components, updated software development file information, the updated STLDD, SDDD, IDD, DBDD, review of updated source code, informal CSC integration test procedures, preliminary Software Test Procedures (STPRs), the CSOM, SUM, CSDM, the updated SPM and the FSM.

1.1.3.6 Computer Software Component (CSC) Integration and Testing

Updated design documentation, source code, object code, and associated listings are entered into the Developmental Configuration for each CSC during Integration and Testing by the contractor. The documents entered into the developmental configuration consist of modified source and object code. A Test Readiness Review (TRR) aimed at checking informal CSC integration test results and the completeness of the STPR for each CSCI is held by the contractor with the objective to review informal test results, formal test procedures, and operation and support documents with the contracting agency. Other documents presented consist of the updated CSOM, the SUM and the CSDM. Successful completion of a TRR indicates that the contracting agency believes that the informal test results and software test procedures provide a satisfactory foundation for proceeding into formal CSCI testing.

1.1.3.7 CSCI Testing

The Product Baseline is comprised of the configuration documents for the CSCI(s) that make up a system. The Software Product Specification (SPS) for a CSCI is entered into the Product Baseline once the contracting agency authenticates the SPS, and the Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA) are successfully completed. The FCA is held to demonstrate to the contracting agency that the CSCI was tested and met the Software Requirements Specification (SRS). Another goal of the FCA is to show that the CSOM, SUM, and CSDM address the needs of the computer system's operation and support. The PCAs purpose is

to demonstrate to the contracting agency that an up-to-date technical description of the CSCI is contained within the SPS. The FCA and PCA for a CSCI may be delayed until the system level if the CSCI requires formal testing upon system level integration. Once the SPS is entered into the baseline, the CSCI Developmental Configuration may cease to exist.

1.1.4 Quality Evaluation

Software Quality is an effort made by the contractor to build quality into software by an evaluation of the products, the proposed methodologies employed to produce the product, and through an assessment process that assures the proposed methodologies were properly followed in the production of the product. To ensure this quality, the contractor needs to define and maintain a set of software standards against which the quality of software is evaluated. Equally important is the design of a standard which defines methodologies and tools that help to determine the quality of software and documentation and the definition of a process to provide quality feedback on identified deficiencies.

Per DOD-STD-2167, a Software Quality Evaluation (SQE) involves the establishment of internal procedures to:

- evaluate the requirements established for the software;
- evaluate the methodologies established and implemented for developing software;
- evaluate the products of the software development process;
- provide feedback and recommendations based on these evaluations that can be used to effect improvements in the software quality; and
- perform corrective action in terms of detecting, reporting and tracking problems with controlled software and documentation.

The means of evaluating the procedures are specified in either the Software Quality Evaluation Plan (SQEP) or the Software Development Plan (SDP). The need for independent individuals with sufficient responsibility, authority, resources, and independence specified in the SQEP or SDP is stressed.

The major SQE activities involve planning, internal reviews, formal reviews and audits, acceptance inspections, installation and checkout, evaluation of subcontractor products, commercially available, reusable and Government furnished software use, preparation of quality records, quality reporting, corrective action system reporting, and quality cost data collection. Each of these activities and a short description is provided in Table 1.1.4-1.

The products delivered as proof that a software quality evaluation was performed consist of : quality records of each quality evaluation performed; quality reports that summarize results and recommendations of quality evaluations performed in preparation for Government reviews; and certification as evidence that each required contract line item was delivered as specified in the contract.

Table 1.1.4-1: DOD-STD-2167 Software Quality Evaluation Activities

ACTIVITIES	DESCRIPTION
Planning	Task planning
Internal Review	Reviews of methodologies proposed in the contractor's planning documents, of software development products and of each software development phase. Internal reviews should be held for all phases: the Software Requirements Analysis Phase, Preliminary Design Phase, Detailed Design Phase, Coding and Unit Testing Phase, CSC Integration and Testing Phase and the CSCI Testing Phase.
Formal Reviews and Audits	Products should be reviewed and audited in preparation for Government reviews.
Acceptance Inspections	The contractor should have all products ready for a Government acceptance inspection.
Installation and Checkout	To make sure that installation and checkout of software complies with subcontract requirements.
Evaluation of Subcontractor Products	To evaluate subcontractor products for completeness, technical adequacy, and compliance with subcontract requirements.
Commercially Available Reusable and Government Furnished Software	Evaluation of the plan to insure that relevant software factors have been considered for commercially available, reusable and Government furnished software.
Preparation of Quality Records	Records of each quality evaluation should be maintained along with the evaluation date, evaluation participants, items or activities reviewed, objectives of the evaluation, all detected problems and recommendations which result from the evaluation.
Quality Reporting	Reports should be written with the results from quality evaluations.
Corrective Action Systems	A Corrective Action System should be implemented for all software and documentation under either Government or contractor control.
Quality Cost Data	Data should be gathered on the cost of detecting and correcting errors in all contractor and Government controlled software and documentation. The actual data collected should be stipulated in either the SQEP or the SDP.

1.1.5 Reserves

1.1.5 Reserves

The need for memory reserves and throughput is stressed by DOD-STD-2167 to emphasize that delivered systems should provide for modifications, maintenance, and enhancement once the system has been delivered. Sizing and timing parameters for each CSCI, to include minimum reserve capacities, are identified and established during the Software Requirements Analysis and the Preliminary Design Phase at which time estimates for parameter values and allowable margins are determined. Throughout the remainder of the software development process, memory size, processing time and reserve capacities undergo continual monitoring and reallocation. Records that reflect actual values are updated during integration and testing when parameter values get compared with overall CSCI sizing, timing and reserve requirements.

1.2 Shortcomings of DOD-STD 2167

Although DOD-STD-2167 is expected to provide better control over software development contracts than its predecessors, there are still some major loose ends which were scheduled to be resolved with the issuance of Revision A. The primary limitations, discussed in the following subsections, include

- Software problems unaddressed by 2167;
- Open issues; and
- Sequential nature of the model.

1.2.1 Software Problems Unaddressed by 2167

DOD-STD-2167 does not resolve all problems addressed by the Joint Logistics Commanders Computer Resources Management Group. The unaddressed problems are extracted from the categories of:

- Software Life Cycle;
- Software Environment; and
- Software Product.

The problem categorization identified is based on the report issued by the DOD Joint Services Task Force, *Report of the DOD Task Force on Software Problems.*[12, Druffel]

DOD-STD-2167 does attempt to resolve the problems stipulated in the DOD Joint Services Task Force report. The report identifies the requirements phase as one of the most vital to the software development process in that requirements analysis and definition was the least defined activity and also the most subject to change. In an effort to remedy inadequate definition, DOD-STD-2167 calls for a Software Specification Requirements (SSR) document which serves as a fixed requirements

baseline and foundation for system design, code and test efforts. In order to prepare the SSR documentation, system requirements must be thoroughly discussed between the contractor and the contracting agency. The objective is to reduce cost and schedule delays which propogate from requirement changes incurred from ill-defined or incompletely specified requirements.

However, freezing requirements does not necessarily provide a solution to problems, particularly where requirements are misunderstood, improperly written, or simply unable to be identified (for example, due to the novelty of the application) at such an early stage in the project. Oftentimes, the contracting agency mistakenly believes that the SSR completely reflects the system needs. The contracting agency's interpretation of what the system will accomplish according to the SSR and the contractor's understanding and description of the system's capabilities may appear similar yet differ in meaning. The discrepancy may remain undiscovered until system deployment, introducing schedule delays and added costs to repair the software.

The domino effect of poorly defined requirements leads to ineffective management, inadequately designed software and major difficulties with product assurance. Proper management of software cannot be realized without adequately defined requirements. Unfortunately, incorrect requirements can often be attributed to the lack of effective communication between management and the contracting agency at the requirements analysis phase, jeopardizing the software development effort from the very start. Vague or incorrect requirements affect the engineer's ability to design, code and test the system. In design, the provision of an acceptable program solution to problems relies on requirements as defined in the specification. Product assurance cannot be guaranteed when the software requirements do not reflect the system's desired capabilities.

Another problem identified by the DOD Task Force deals with the transition of software, such as from exploratory research to engineering development. One software methodology currently under research and investigation is the use of rapid prototyping as an effective means to define software requirements. This concept of recycling from requirements through system design or implementation and testing in order to successively refine requirements is not adequately supported by DOD-STD-2167.

1.2.2 Open Issues and Revision A

At the time DOD-STD-2167 was published on 5 June 1985, there were a number of open issues identified and agreed to by government and industry representatives that remained to be fixed in a future revision to the standard. It was agreed that there should be a continuing effort within the DOD to develop a Revision A which resolved these issues.

The open items included a set of primary and secondary issues that were deemed important to resolve but it was decided that the resolution process should not hold up the publication of the basic version (DOD-STD-2167, 5 June 1985). Primary issues included tailoring, Ada[1] compatability, and systems interfaces among others. Secondary issues included interface to quality assurance, documentation fragmentation, design methodologies and others. The issues and historical perspective of the issue resolution process are documented in *CODSIA Task Group 21-83 Report on the DOD-STD-2167 (SDS) Package Coordination Review*, dated December 5, 1985.

[1] Ada is a registered trademark of the U.S. Government (Ada Joint Program Office)

1.2.3 Sequential Nature of 2167

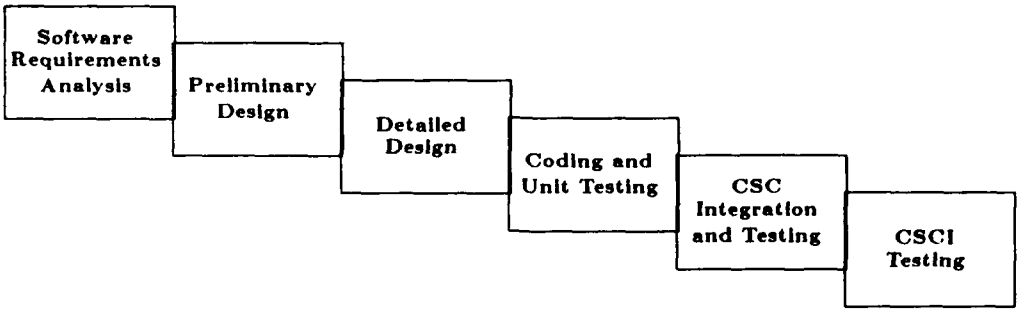


Figure 1-3: Waterfall Software Development Model

The publication of the Draft Revison A of 2167 on 1 April 1987, is the culmination of the rewrite process aimed at resolving the primary and secondary issues. The current on-going review process indicates there may be some disagreement between government and industry reviewers as to whether the issues have been adequately addressed. It's up to the future to determine the outcome of the issue resolution process.

1.2.3 Sequential Nature of 2167

As previously noted, DOD-STD-2167 is based on the waterfall model approach to software development (see figure 1-3). The waterfall model defines separate and distinct phases of software development which, when performed in a top-down structured manner, claim to result in a "soundly designed" software product. The waterfall model has historically been the accepted approach to developing software within the DOD software standards arena.

At the completion of each phase, certain milestones, such as reviews, baselines and completed documentation, should have been accomplished. As such, the waterfall approach is conducive to providing managerial visibility and control into the software development process. The milestones also provide a means of verification/validation for each particular phase which signals the start of the succeeding phase.

The fundamental problem with the waterfall model, as depicted, is the sequential nature of the model and the implication that subsequent activities cannot be performed until previous activities have been completed. An example of this is that design should not be allowed to start until requirements have been completely defined. This is not the case, nor was it ever the intent of the DOD-STD-2167 waterfall model. Instead, the model was meant to depict a customer to developer interface model, and not a process model that the developer would follow in producing the software. A process model would show definite overlap even to the point that coding may be underway during requirements analysis.

1.3 Evolution in the Conventional Software Development Process

1.3.1 Recognition of Prototyping

The concept of prototyping is gaining recognition in the conventional software development environment. The following subsections define the philosophy behind prototyping, and how it can be united with the classical waterfall model.

1.3.1.1 Basic Concepts of Prototyping

Prototyping involves the development of a subset of an entire system which is used to demonstrate the proposed system's functionality to the end-user. Once the users have the opportunity to work with the prototype, they can provide helpful feedback to the designers. The suggestions made by the users can be implemented into a successive prototype, which can be tested by the users again with the goal of obtaining further suggestions. This process is repeated in an iterative fashion until the users are satisfied with the prototype's functional performance.

1.3.1.2 Strengths of Prototyping

The prototyping philosophy demands a high level of user involvement. In turn, heavy user involvement increases the likelihood of system success. By seeing their ideas implemented in the prototype, the users begin to develop a sense of ownership of the system which, in turn, facilitates acceptance of the final product. In addition, the users who worked on the prototype can assume responsibility for training other potential users in the operation of the system.

Another strength of prototyping is its ability to clarify poorly defined requirements. This is achieved through implementing portions of the system and allowing the users to work with it and suggest design changes. The concept of prototyping is extremely useful in terms of increasing the communication level between the analysts and the users. The iterative nature of prototyping aids in flushing out many of the hidden or poorly defined requirements early in the life cycle. This process greatly reduces the risk of costly unforeseen design changes which typically occur late in the life cycle.

Prototyping also provides a mechanism in which potential solutions to the problem can be identified and tested. This allows the designers to determine which solutions are most feasible. When a solution is tested unsuccessfully, it can be discarded and the focus redirected toward finding a more efficient and workable solution. The prototype can also help evaluate the verification and validation issues. Namely, one can use the prototype to answer the questions, "Are we building the right system?" and "Are we building the system right?". When these questions can be answered affirmatively, the system is considered to be verified and validated.

Another benefit of prototyping is the ability to incrementally build the requirements of the system. This, along with the above mentioned abilities, enables the designers to evolve the system specifications into fully understood, clearly defined, workable, and robust system requirements. With all

these critical issues being addressed earlier in the life cycle, the risk associated with developing the system can be greatly reduced.

1.3.1.3 Proper Perspectives

A prototype is intended to model parts of a system; it is not meant to be a complete version of the system. Generally, the user interface components of a system are included in the prototype. In addition, the most technically complex portions of the system should be prototyped to assess the feasibility of the approach early in the development process. Problems can arise from the reluctance of managers to throw away software. This can pressure designers to evolve the prototype into a final product rather than using the prototype to specify requirements and then formally design the final system. In some small systems, using the prototype as part or all of the final product does not present problems. In larger systems, however, this approach could significantly complicate maintenance of the final product.

1.3.1.4 Incorporating Prototyping into the Waterfall Approach

Prototyping should be performed concurrently with the software requirements analysis phase to generate more clear and accurate system requirements. Once the users are satisfied with the prototype's performance, system definition is complete and the project can make the transition into the remaining phases of the waterfall model. Figure 1-4 illustrates the concept of incorporating the prototyping concept into the waterfall model.

1.3.2 Use of Off-the-Shelf Software

Within the DOD, there is a definite thrust to capitalize on software already in the commercial marketplace as long as it meets the needs of the system under development. DOD-STD-2167, in particular, advocates the use of off-the-shelf (OTS) software but requires that its use be approved prior to incorporation into a system. Many developers propose OTS software as part of system development with the result that numerous fielded systems contain these elements.

As we progress into the Ada age, the concept of reusability is being espoused as a means for improving productivity. It is reasonable to expect that some of the reusable software will be OTS in nature such that it is produced for commercial sale to programs being developed for DOD. The primary concern on the part of DOD projects are rights issues and the ability to have a supported product once a system is fielded. It is a management decision to address these issues on every application of OTS software and to conduct the proper economic analysis to substantiate the decision.

1.3.3 Compatibility with AI Software Development

The previous concepts pertaining to the evolution of the conventional software development process fit in quite well with the practice of developing knowledge based systems (KBS). Namely, prototyping is an integral part of KBS development, as discussed in Volume I. Because AI techniques are

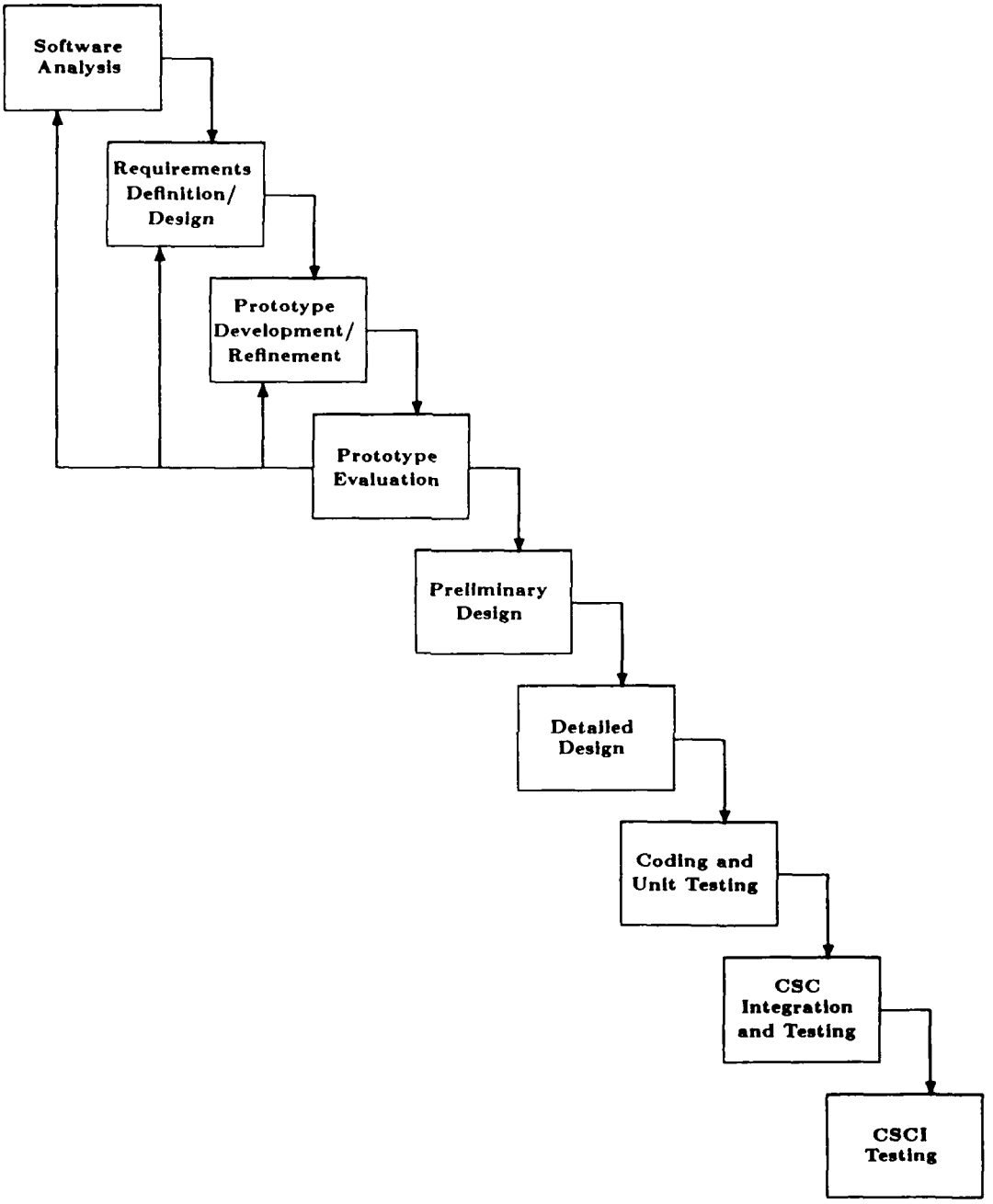


Figure 1-4: The Prototype Life Cycle Model

1.3.3 Compatibility with AI Software Development

generally applied to ill-structured problems, prototyping is the means to a better understanding of the problem itself. As the problem definition becomes increasingly clear, the associated requirements can be more accurately stated. Furthermore, well-defined requirements greatly enhance the probability of a successfully built system.

Prototyping is also used early in a project to demonstrate the feasibility of a given system or proposed approach. It is also an invaluable mechanism in terms of acquiring and implementing domain expert knowledge.

With regard to reusable or "off-the-shelf" tools, a KBS can generally be described as consisting of 3 major components:

- Knowledge base;
- Inference engine; and
- User interface.

The knowledge base, which contains all relevant domain knowledge, is a complex entity that must be uniquely developed for each application. However, the inference engine, which navigates the system through the knowledge base in search of an acceptable solution, is a fairly standard mechanism. Consequently, inference engines of several varieties are commercially available.

Shells are more encompassing tools which generally include an inference engine as well as a graphically oriented user interface. There are many KBS shells on the market today suited for a wide variety of machine architectures ranging from personal computers to large main frames. In an effort to save resources and/or allow the developers to focus on the critically important generation of a system's knowledge base, the use of "off-the-shelf" tools is quite common and certainly available in the technological marketplace.

SECTION 2

Properties of a KBS Development Model

2.1 Provisions

Prior to defining the composition of a KBS development model, it is important to consider the goals to be met. Ideally, we expect such a model to provide both the DOD and its contractors with a mechanism for successfully developing systems. In this context, a "successful system" is one that is functionally satisfactory to the DOD, and also complies with the previously established budget and schedule criteria.

With these goals in mind, a KBS development model should provide the user with the following features:

- Visibility;
- Control;
- Flexibility; and
- Compatability.

Note that these features should be inherent to a general AI software development model. However, due to the nature of the case study data presented in Volume I, we are specifically addressing KBS developments.

The following paragraphs expand upon the four features listed above.

2.1.1 Visibility

A project with a large software investment requires a considerable amount of attention to management and control of the software development process. Large software projects are typically broken down into manageable and modular phases to provide increased visibility into the development process and to provide management with a more efficient and effective means of controlling this process. A phased approach to software development supports the proper allocation and management of people, product identification and the definition of essential activities, baselines and reviews.

Specific segments of the software development process for which visibility is particularly important include the following:

- Progress tracking;

2.1.1 Visibility

- Requirements definition;
- Design and implementation;
- Documentation preparation;
- Product quality; and
- Software change control.

The manner in which visibility pertains to each of the above segments is discussed below.

The progress of a software project is often based on management's perspective and the magnitude of identifiable products to be delivered. Management's concerns revolve around the availability of current status information concerning schedule, cost, and resource expenditures. The development of a KBS differs from a conventional software system in that an *iterative* approach, including the development of prototypes, is used. Scheduled technical and management reviews at predefined intervals provide a means for management to observe the system's ability to perform as required, as well as information to identify potential risks or areas of uncertainty. In addition, the prototype reviews also provide a means to determine cost and resource needs for the next iteration. Consequently, updated versions of the requirements can be provided as a product of each prototype review. This review process is necessary to track development progress.

Successful system implementations depend upon the correctness and completeness of expected requirements. Because problems suited to KBS resolution are inherently ill-defined at the outset, the dynamic requirements definition process demands high visibility. The progressive refinement of requirements based on successive prototypes involves an effort intended to remove inappropriate goals and to add new requirements as they are identified.

The need for visibility into software design and implementation surfaces particularly with respect to maintenance of the software in its deployed environment. The level of difficulty involved with software maintenance increases substantially with the lack of proper design and code documentation. In-process design reviews provide both a means of tracking progress and a means for assessing design documentation completeness in support of those who will ultimately be responsible for maintenance.

Product quality is also based on visibility into the software development process. The concept of *building in* quality is critical. Prototypes provide direct visibility into a product while simultaneously providing an opportunity for isolating and recognizing testing needs. As will be discussed in Section 2.2.4, an informal testing process occurs throughout the prototype development activity. With each prototype, ideas pertinent to a formal evaluation may be documented for inclusion in a test plan.

Visibility into software changes as they relate to the requirements definition, design and/or implementation is essential to the overall effort. There must be traceability to identify the source of a problem as well as accountability to ensure that changes are effected.

2.1.2 Control

Control, another desirable characteristic of sound engineering practices, allows management (customer and contractor) to create and maintain a functional system based on certain standards and regulations. Software control appears in:

- the software development effort;
- project changes;
- contract changes;
- document and report generation;
- software quality; and
- reviews and audits.

A discussion on each type of control follows.

Management's ability to effectively control the software effort is proven by the delivery of a working system. With iteratively developed systems, the responsibility includes regular delivery of planned prototypes. Some of the major concerns of management involve the exertion of control over the schedule and budget, as well as reporting status and financial information to upper management and the contracting agency.

An essential management function is the control over changes in the developing system. Management needs to determine what to control (a foundation or baseline), the kinds of changes to be controlled, and the mechanism for effecting that control.

A means to organize and assess contract changes is another software control issue. Contract changes should be examined by a competent technical group. Estimates on the cost to implement a change should be determined. Formal approval by the customer is the final factor determining whether a contract change should be implemented.

Document control contributes vitally to the success of a project. Therefore, a need exists to create, update and maintain documents. Proper organization of a document library is generally required.

The quality of a system is based to some degree on the ability of a system to meet stipulated customer requirements. However, the system's functionality also depends upon the system's ability to accomodate future needs. In the extreme, it may be desirable to advocate always building a more general system than required. One way for management to assess the final quality of a system is to understand and determine the quality of the baseline design well before acceptance time.

Finally, formalized reviews and audits provide a means for measuring progress based on a well-defined set of criteria. Each review or audit can be designed to present information appropriate to a program phase or stage. The adequacy of the information can be judged and placed under control when the related criteria have been satisfied. As an example, requirements reviews would examine requirements completeness and when found to be acceptable, placed under control. In a like manner, design and code can be judged with the result of establishing control over them. Reviews and audits provide one of the more important control mechanisms in a development project.

2.1.3 Flexibility

A KBS software development model must satisfy the acquisition needs of the contracting agency. At the same time, it must provide flexibility so as not to unreasonably constrain or inhibit the software engineering process. Specifically, the model must be tailorable and supportive of various KBS application requirements, such as real time needs, mission critical environment, etc. The degree to which the model is applied should primarily be determined as a function of the criticality and support requirements of the subject system.

The specific development environment proposed by the contractor should not be unduly restricted. Namely, a contractor should be allowed to prototype on the hardware of his choice so long as the software can be easily transitioned to the target machine or on-site hardware. Use of shells or other tools should also be acceptable given the provision that they are compatible with the target hardware and provided they have reached some level of maturity such that one can trust their use. The model should also accomodate various knowledge representation implementation techniques such as rules, frames, constraints and the like. Specific restrictions are not advisable in today's KBS environment where technological growth is rapidly increasing.

In terms of language selection, prototyping activities should not be arbitrarily constrained. Namely, a contractor should have the latitude to prototype in any "reasonable" language as long as implementation of the system in the target language will be straightforward. Of course, the contracting agency should be able to substantiate the rationale behind the choice of language during the prototyping or system definition phase. Although flexibility in language selection is advocated at this point, it is not unreasonable to expect that DOD will in the future explore the standardization of a common AI language.

Flexibility also implies that the model should not inhibit growth of a larger system. In addition, the model should be able to accomodate future technology standards. For example, consider the standardization of COMMON LISP as a prototyping language.

A KBS software development model should also provide flexibility in terms of contracting mechanisms. The number of prototypes generated and the time spent in the system definition phase may be difficult to predict at the outset.

2.1.4 Compatibility

2.1.4.1 System Development

It must be recognized that KBS's may simply be components of a much larger system. Consequently, the development of a model must be able to be tied to the system engineering process. This process is one which starts out by identifying the component parts of a system through system engineering trade-off studies and, once defined, attempts to ensure all components do in fact work or interface with each other. Compatability between an AI system component and the more conventional hardware and software components is essential to the overall system development process.

Compatability is also necessary during the system integration process. At some point during the development cycle for a system, components need to be integrated and tested as a total system. It

is important that this interplay between system components and the KBS model exists and that there not be incompatabilities that inhibit the development process.

2.1.4.2 Conventional Software Development

Many large system developments include a combination of knowledge-based as well as conventional software components. In fact, PAR Government Systems Corporation has indicated that, of the intelligent decision aids they have built within the past five years, the knowledge-based software components comprised approximately 30% of the entire system. The rest of the systems were made up of a combination of databases and conventional support software.

As a consequence, an adequate KBS development model must mesh well with its conventional counterpart, DOD-STD-2167. Although KBS are not constructed in a top down manner, the development process can still be described in terms of phases. These phases should be identified so as to provide similiar checkpoints and milestones as the concurrent conventional effort if we are to maintain compatability. In this way, the project as a whole can be effectively and consistently managed with visibility and control over both types of software.

In addition, the KBS model needs to provide hooks to the conventional software from a communications standpoint. Specifically, interface review meetings and documentation must be called out to ensure proper development at the system level.

2.2 Composition

The composition of a KBS software development model must reflect the actual process employed by the developers in the expert system community as well as satisfy the needs of DOD in terms of contracting for such systems. In general, KBS developers have had little control imposed on their efforts. On the other hand, DOD software contracts are tightly controlled, using a standard such as 2167 to define activities, products, reviews and baselines. Consequently, a KBS model that is best suited to both parties might be characterized at the extreme as a tender balance between bipolar forces.

The following subsections generally describe the KBS software development process and identify pertinent DOD concerns, such as documentation needs, configuration management and testing. These concerns are reflected through various DOD policies governing the acquisition of software and through the current DOD standard for software development, DOD-STD-2167. The discussion provides the basis for the model definition presented in Section 3.

2.2.1 Activities Identification

The development of a KBS appears to follow three major processes:

- a system definition stage;
- a system implementation stage; and

2.2.1 Activities Identification

- a system operation stage.

Each stage is examined in the following three subsections. Figure 2–1 provides an overview of the major activities identified in each stage.

2.2.1.1 System Definition

The system definition stage is characterized by high iteration as incrementally developed software is proven acceptable in the form of regularly demonstrated prototypes. The process involves management planning closely intertwined with knowledge engineering as a means to: plan the software engineering strategy; find and extract information from a domain expert; and represent the information in a knowledge base. A high level description of the process follows. Refer to Volume I for a more detailed accounting of specific aspects of knowledge engineering.

The management plan accounts for the identification of problems, participants, resources, goals and approaches. The initial identification of a problem may not necessarily represent an appropriate expert system task or may include a scope that is too complex to implement in the first version. This may result in an effort that is plagued with unsolvable design problems, that costs more than it saves or that is unacceptable to users. Therefore, precautions should be undertaken to insure that the problem reflects a true 'expert system' task that is reduced to a manageable, but reputable size. Some key areas where expert system development might help include situations where[17, Harmon and King]:

- solutions to problems are based on subjective, human expertise;
- key individuals are in short supply;
- large team efforts are needed to solve small performance tasks;
- performance degradation occurs because one individual cannot grasp the entire issue;
- a system performing a portion of the total task would be useful;
- large discrepancies exist between the best and worst performers;
- incorrect or nonoptimal results can be tolerated;
- a lack of human resources compromises corporate goals; and
- competitors can resolve a given task better.

In contrast, problems that are stable, and lend themselves to numerical solutions should be developed through conventional methods.

Upon identification, participants in the expert system development process (a selected group of knowledge engineers, experts, users and customers) assess and subdivide the problem until a formidable, but manageable problem is defined. Qualifications used to select an expert must be specified. Some criteria for a choice are the expert's general acclaim, reputation, ability to

System Definition

- Define problem
- Plan software engineering strategy
- Prototyping
  - Knowledge Acquisition
  - Knowledge Representation
  - Informal Testing

System Implementation

- Coding
- Knowledge Base Refinement
- Informal/Formal Testing
- Customer Acceptance

System Operation

- Delivery
- Installation
- Maintenance

Figure 2-1: Top Level View of KBS Software Development

communicate, availability, and at a more objective level, perhaps, the expert's level of training, years of experience and position in the company. Once the problem and participants are identified, resources to support the system are determined. Some resources include time commitments from experts and knowledge engineers as well as a definition of the necessary hardware to include knowledge engineering tools designed for rapid prototyping of expert systems. The system goals and approaches refer to management's decision on how to control the knowledge engineering process, including a schedule for prototype reviews. Management also determines how and where to implement configuration management, defines what kind of quality controls are needed and specifies how the system can best be supported and maintained.

The knowledge engineering process involves knowledge acquisition and representation with the ultimate goal of producing a prototype. Acquisition of knowledge is based on the interaction between the knowledge engineer, who examines different techniques on how to effectively extract information from the domain expert. The effort revolves around the establishment of a method to understand the problem and to characterize the data and possible problem solutions in the knowledge base.

Knowledge representation is closely intertwined with the knowledge acquisition process. Throughout knowledge based system definition, the two steps are revisited repeatedly in an effort to define and then refine the knowledge base. During knowledge representation, the knowledge engineer attempts to map the knowledge acquired into a formal representation. The nature of the domain knowledge will likely affect the decisions relative to purchasing an expert system shell, a tool which facilitates the knowledge representation process. Once the information is formalized, informal testing by means of a prototype review which may include members of the user community, maintenance organization and the contracting agency allows for a validation of the data and performance.

The prototype is successively refined after each demonstration or review. In some cases, the prototype may even be thrown away. Nonetheless, each prototype fulfills a purpose in identifying errors such as invalid knowledge, incorrect inference rules and/or inaccurate control strategies. Another cycle through knowledge acquisition and representation corrects the errors, allows for necessary enhancements, brings up other problems to resolve and helps the knowledge engineer determine the desirable performance qualities of the system. At the end of each prototype review, requirements and test plans should be refined or updated.

2.2.1.2 System Implementation

With the problem well defined, the implementation process begins. In this process the "production-quality" system is implemented in the target environment, which may be different from the prototyping environment. In some instances, the prototype may have been developed on the target hardware, making it possible to evolve the prototype into the operational system. The decision to evolve the system depends on the properties of the prototype (i.e. the software may not be well structured or maintainable). In addition, the feasibility of evolving the prototype depends on the level of modifications necessary in order to satisfy the system's requirements. Namely, the implemented system will need to be more robust than the prototype in terms of cases covered and the ability to reason with uncertainty. These kinds of enhancements will effect the knowledge base and the inferencing mechanism.

In addition, error handling, which is not a major issue during system definition, must be considered during system implementation. Efficiency concerns, in terms of memory utilization and run time response, present another set of implementation criteria that must be addressed.

If the target environment is incompatible with the prototype, the system must be reimplemented.

The system implementation process is somewhat iterative by nature, although not as iterative as the system definition process. As the system is implemented, the need for knowledge base refinements become apparent. As the knowledge base changes, the system undergoes informal testing which can uncover deficiencies that need to be re-implemented. This process continues until the system is ready for formal testing and customer acceptance.

In systems where an interface to conventional software exists, system integration must occur. This effort can be accomplished near the end of formal testing. With an operational and integrated system built, the project can transition into the system operation process.

2.2.1.3 System Operation

The bridge to system operation occurs with formal customer acceptance of the final system based on the test approach established during the system implementation stage (see Section 2.2.4). The main goals of system operation revolve around delivery and installation of the final system, and the establishment of firm maintenance capabilities to enhance the system or correct errors and deficiencies. Enhancements to the system may involve a pass through the highly iterative system definition phase to acquire and represent the new knowledge and/or a pass through the implementation phase to update the system. For changes to the delivered system, configuration management and quality control should continue to be active and updates to documentation and extensive testing provided to establish a sound system for the next delivery.

2.2.2 Documentation Needs

Along with the delivered code, substantive software documentation is an essential component of a high quality, supportable system. In addition to defining the software, documentation provides a means for customer/developer communication throughout system development. Scheduled releases of pertinent documentation reduce the risk of building an unacceptable system.

A KBS development model must require the submittal of a specified set of documents to support the delivered system. The following subsections describe a number of documents designed to facilitate maintenance of a KBS. These documents are also conducive to *building the right system the first time*.

2.2.2.1 Management Plans

Management plans are effective when written during the project proposal stage or early into the software development effort. From a management perspective, important issues to address are:

- how/when the software will be developed;

2.2.2 Documentation Needs

- what software environment will be used (hardware/software);
- how the domain expert(s) will be chosen and how much time he/she will be committing;
- what coding standards will be followed;
- what and when documents will be delivered;
- what and when reviews will be held;
- what configuration procedures will be employed;
- what software quality measures will be followed;
- what testing criteria will be employed; and
- program schedule and staff organization.

Management plans can be useful working documents for both the customer and the developer. By describing the software development methodology, the customer has visibility into the evolving product. By knowing the proposed strategy *up front*, feedback and suggested changes can be offered by the customer.

Management plans also provide the customer with control mechanisms. Namely, the proposed schedule allows the tracking of progress by attendance at major reviews and evaluation of product deliveries against the stated delivery dates. The likelihood of developing a successful system is enhanced by allowing the customer to become involved.

At the same time, management plans are also of benefit to the developer. In the process of developing these plans, the project manager has defined the means by which the project will be run and internal progress measured. In addition, the plan defines the development environment and the team needed to implement the system and reach project goals. A system would surely be doomed for failure without planning of this nature in the early stages of a project.

2.2.2.2 Engineering Documentation

Engineering documentation provides a roadmap for the system to be built by specifying system requirements and providing designs and system implementation details which greatly facilitate operation and support efforts.

The following information should be included in the specified set of deliverable software documentation for a given project.

2.2.2.2.1 Requirements Definition At the start of a project, an initial set of requirements establishing the broad goals of the system is generated. Because problems suited to KBS resolution are inherently ill-defined, the requirements initially listed may be incomplete or even inaccurate. Nonetheless, they offer a starting point from which the system definition process can begin.

During the system definition stage, requirements are refined as prototype generation provides insight to the problem. At the completion of system definition, the final set of software requirements, presumably complete and accurate, must be documented. Software interface requirements should also be included in the documentation.

2.2.2.2.2 Functional Design Descriptions Following the iterative system definition stage, the developers should have insight into *how* they propose to solve the problem at hand. Specifically, the developers should know how the solution breaks down in terms of which elements of the problem are best solved by hardware, KBS techniques and conventional software techniques. At a functional level, these components as well as interfaces can be defined. The result is a document that presents a conceptual design of the system.

Knowledge acquisition and representation are key activities of the system definition process. Throughout prototype generation, ideas concerning the best and most natural way to represent domain knowledge are developed (i.e. devising the vocabulary or epistemology in terms of which the "rules" are written). At the end of system definition, the knowledge engineers should have strong perceptions concerning knowledge representation and inferencing mechanisms. A functional design document could also present this culmination of ideas including justification for the recommended knowledge base and inference engine designs. Presenting the *why's* behind the design is useful for several reasons:

- The risks associated with the chosen approach as well as the discarded approach(es) are identified for consideration by the customer;
- The information provides the customer with a better technical understanding of the approach recommended for system implementation; and
- Knowing *why* decisions were made often facilitates the maintenance/support cycle of a system.

2.2.2.2.3 Product Specification During system implementation, a software description document should be written to annotate the code at both the *module and unit* levels. This document is considered to be an *as-built* description of the system.

A product specification should also completely describe the knowledge base including the knowledge source(s). Suggested paragraphs include a description of the knowledge base in terms of its contents, structure/architecture and relationships between knowledge and/or other software components. For example, if one were to change a rule, what other software entities would be affected?

The knowledge representation format, such as rule or frame formats, should be defined.

A complete description of the inference engine, including inheritance properties, constraints, inexact reasoning, truth maintenance, explanation capabilities and the like should also be part and parcel to the product specification. Because the explanation capabilities of a system trace the reasoning paths that have been followed in reaching a conclusion(s), they play an important role in system maintenance and support.

2.2.2.3 Test Documents

Following the system definition stage, the project team should specify their plans for informally and formally testing the system. Because KBS are nondeterministic by nature, the criteria for evaluating whether a response is acceptable or unacceptable must be established a priori.

2.2.3 Configuration Management

Based on the final requirements specification and experiences gained through prototyping, the developers should be able to define a *class* of test cases suited to system evaluation. For example, consider a system that diagnoses upper respiratory disorders. A class of test cases might include the plan to check for the diagnosis of a common cold, an ear infection, strep throat and tonsilitis. This type of information can be included in the testing section of an updated software development plan.

Nearing the end of system implementation, a test description document could be used to specify particular test cases. For example, from the class of test cases stated above, specific inputs, intended to lead to the kinds of maladies mentioned, can be established.

Lastly, a document recording the testing results for all cases specified should be required.

2.2.2.4 Operational and Support Documents

Prior to installation, the contractor should deliver documents pertinent to operation and support of the system. In terms of operation, manuals that define the procedures for operating the computer system, as well as the software, should be required.

Support documentation should also be submitted to facilitate maintenance of the delivered computer system and the software.

2.2.3 Configuration Management

An important aspect of any software development effort is the management of software changes. Configuration management refers to the concept of controlling changes to a previously defined program to demonstrate that the original definition of the software was modified appropriately. Generally, the unit of work controlled is a subsystem referred to as a Computer Program Configuration Item (CPCI) or Computer Software Configuration Item (CSCI). As previously described, the DOD has established three major baselines to govern the control of software throughout the life cycle:

- the functional baseline: produced during Pre-Software Development;
- the allocated baseline: which occurs after Software Requirements Analysis; and
- the product baseline: which follows CSCI Testing.

In addition, internal configuration management of software should be maintained by the contractor. The configuration management for expert systems follows the same general pattern of control stipulated in the standard with a major difference regarding allocated baselines.

Given the three general steps identified as an approach to expert system building: system definition, system implementation and system operation, the functional baseline falls at the outset of system definition prior to prototype development; the allocated baseline may occur at the end of system definition and the product baseline appears at the end of testing and reviews in system implementation. The iterative nature of the system definition stage complicates the configuration

management process with regard to the allocated baseline. Requirements are not established and fixed at the outset as in conventional software development. They are progressively defined and refined through successive prototype development until the contracting agency, satisfied with the defined requirements, finalizes the review process and agrees that the system can be implemented based on the requirements. Each prototype represents a set of requirements which may serve as the foundation for the final implemented system or serve as a stepping stone to the next set of requirements. Therefore, a method to track software changes between prototypes must be available. One solution is to maintain an evolving allocated baseline that exerts a predetermined level of control over each prototype. Since prototype changes are often stimulated by testing new cases, it may be wise to tie changes in the "rules" to the cases that triggered them. Once the final prototype is determined, the allocated baseline can be finalized. An evolving baseline provides a solution to the situation where a project meets with unexpected circumstances such as a termination due to insufficient funds or a decision to base the implemented system on the previous prototype because of financial limitations, etc. Internal configuration management remains an important aspect throughout system implementation and until the product baseline is established.

Another area of configuration management concern is status accounting. A means to track information in a document or in a file is essential to a project's ability to maintain a status record on the project at all times. The data should be collected as part of the configuration management process.

2.2.4 Testing Approaches

A KBS software development model must provide an approach for testing system accuracy and utility. The testing and evaluation process can then be used as the basis for releasing the system to a fielded or operational environment.

Traditionally, knowledge based systems developed to date have undergone only an informal evaluation process with few systems actually fielded. As a system is being developed and successive prototypes built, the domain expert is repeatedly conducting informal evaluations. Static evaluations generally occur wherein the system knowledge base is checked for consistency and completeness relative to the expert's own domain knowledge. Dynamic evaluation may also occur wherein the system's line of reasoning on a specific test case is compared with the expert's own reasoning process. The knowledge engineer works closely with the domain expert to refine the knowledge base with the evolution of each prototype system. Figure 2-1 presented a broad view of the KBS development process. It is during the *system definition* phase that the notion of informal testing occurs as the prototypes evolve.

Once system definition has reached a mature state, the requirements for the system begin to stabilize. At this stage, the system goals can be established with a high level of confidence. The goals should include the type of assistance required by the system (i.e. intelligent assistant vs. an autonomous system) as well as the minimally acceptable accuracy level. Given the system goals, the domain expert and knowledge engineer can begin to define a class of test cases to which the completed system should correctly respond.

Following system definition, the next major stage in the development process is called *system implementation* as shown in Figure 2-1. It is during this phase that a system can be implemented

according to the most recently postulated set of requirements or goals. The system as implemented may differ from the latest prototype in terms of language, target machine, efficiency considerations and the like. Prior to releasing the system for operational use, it is likely that the customer will require a formal testing procedure.

In addition to customer acceptance testing, wherein the system must give a rigorous accounting of itself, there are several other reasons that support a formal testing process. Namely, user confidence in the system is largely based on credibility which can be demonstrated via formal test procedures. In addition, for systems where maximum accuracy is critical, a formal test process can be used to flesh out errors which may not be uncovered during informal reviews conducted by the domain expert and knowledge engineer. Lastly, legal concerns may surface if a system has not been thoroughly tested with documented results.

A formal test approach that could be used in the validation and verification of a knowledge based system is shown in Figure 2-2. As indicated, at the end of the system definition phase, a *class* of test cases can be developed by the domain expert and knowledge engineer on the basis of the finalized list of system goals.

The specific steps that may be involved in formally evaluating an implemented system are shown in Figure 2-2. The first step in the formal process may be to introduce a sense of objectivity by obtaining an independent expert(s) to test the system. Because domain experts are generally difficult to find and usually extremely busy people, whether or not to seek an independent expert may depend on the accuracy and/or security requirements of a given system. If a decision is made to procure an independent expert(s), one should review the qualifications for defining an expert that have already been established as part of the management planning activity. One should also keep in mind that the independent expert(s) should use roughly the same reasoning approach as the original domain expert to avoid total chaos during the evaluation process. If an independent expert(s) has been contracted, he/she can begin to derive a set of *specific* test cases from the *class* of test cases previously generated. Otherwise, the domain expert proceeds with this task as well as the other tasks described to be the responsibility of the independent expert in the following paragraphs. The specific test cases can then be incorporated in the test description document. Once complete, the customer should review and approve the test descriptions which will form the basis of the acceptance testing activity.

When the system has been fully implemented, actual testing can begin. The independent expert(s) should initially conduct static evaluation to compare the system knowledge base with his/her own and identify any problem areas. Dynamic evaluation can proceed next whereby the system's line of reasoning and conclusions on each specific test case are compared with the independent expert(s) own thoughts. Differences that may arise between the domain and independent experts must be resolved. If applicable, errors should be corrected and the system retested. As a consequence, the static and dynamic evaluation steps may be revisited a number of times. When a resolution cannot be reached amongst the experts, the decision may need to percolate to a managerial level wherein the arguments from both sides are weighed, especially as they pertain to the system goals.

Documentation of the test results should be an ongoing activity. Once differences between the experts have been resolved and the customer agrees that the system meets the predefined goals, he/she should sign off on the system. At this point, the system can be delivered.

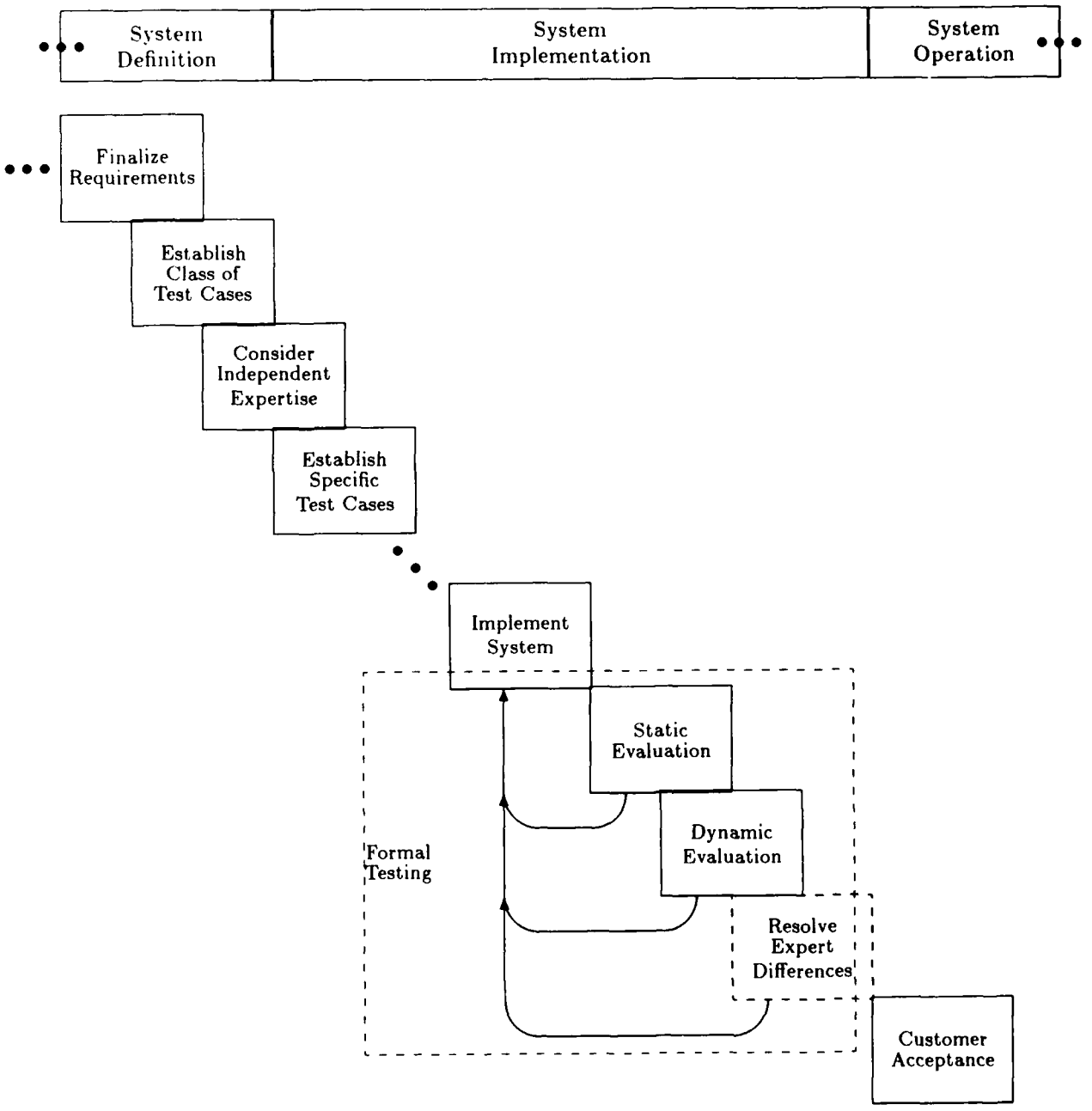


Figure 2-2: KBS Formal Test Approach

2.2.5 Quality Evaluation

Software quality evaluation is a continual process which occurs throughout software development to help organizations minimize the impact of problems discovered late in a development effort. As DeMarco states: "the major determinants of quality are for the most part already in the software before testing begins." [10, DeMarco] Although quality can be defined in a number of different ways, some of the purposes for quality metrics reside in ensuring that delivered software complies with contract requirements; providing management with visibility into software that is periodically assessed via reviews and audits; and applying metrics to software products throughout a software life cycle to help with identification of the final product's quality level or the detection of quality-related problems. The definition of software quality is open to many interpretations and is essentially defined by the person(s) who must determine the level of quality for a particular system. Nonetheless, some of the major quality concerns seem to revolve around the following three factors:

- **performance quality factors** which deal with the software's ability to perform or to degrade gracefully when faced with error occurences that affect the system's functionality. Some quality related areas included in the evaluation may be:
  - usability;
  - reliability;
  - survivability;
  - efficiency; and
  - integrity.
- **design factors** which refer to the ability of software to work properly and as determined by the requirements. Areas considered to affect the design quality of a system may include:
  - correctness;
  - maintainability; and
  - verifiability.
- **adaptation factors** deal with the software capacity to satisfy requirements and to allow for extending or expanding capabilities and/or adaption to other environments or applications. Specific concerns are then:
  - expandability;
  - flexibility;
  - interoperability;
  - portability; and
  - reusability.

Each of the major factors and areas identified can be interpreted in any number of ways to accomodate a variety of different quality assurance programs. The following section attempts to identify key areas where quality should be examined throughout expert system development with respect to performance, design and adaptation factors.

2.2.5 Quality Evaluation

One of the major functions of an expert system is to perform in the capacity of an expert who has shared his/her knowledge with an engineer who incorporates the information into the knowledge base. To provide the performance capabilities necessary to the survival of the system, performance quality measures in terms of usability, reliability, survivability, efficiency and integrity should be determined.

A system's success may depend upon the user's acceptance and use of the final deployed system. An advantage of KBS development revolves around the fact that the system is continuously reviewed and updated according to the user's response and expectations. Therefore, quality in the system relies on the ability to accurately assess and determine user needs and build each prototype with efficiency and usability as a major objective.

The system's reliability may be based on each expert's ability to adequately relay and test information together with the knowledge engineer's talent in designing and integrating the knowledge base. Therefore, proper selection of the expert(s) and engineer(s) can affect the system's ability to reflect quality in the information process.

An aspect of survivability is the system's ability to degrade gracefully. Many expert systems rely on continual updates to the domain knowledge. Therefore, expert systems should accomodate some type of modification and maintenance function to allow for planned quality improvements.

Another aspect of efficiency relates to the system's ability to appropriately reflect the expert's knowledge. Continual informal quality evaluation of the knowledge base from each prototype can be done by a second domain expert who should be chosen for consistency in approach to the primary domain expert. The expert should observe the knowledge acquisition process and objectively qualify the information content and the inference engine according to a predetermined standard.

The system's integrity depends on the fact that only key experts will exercise the right to update the domain knowledge. Since the information in the knowledge base reflects a chosen expert's knowledge, the person(s) who have maintenance access to the information should be limited to professional individuals qualified for the task.

The incremental approach to expert system building, *build a little, test a little*, introduces an additional consideration when determining the desirable design quality factors in areas such as correctness, maintainability and verifiability. The correctness of a system may be based upon the software's ability to reflect and represent system specifications. Since requirements are continuously rewritten and demonstrated through a prototype that represents the system's capabilities, the system's correctness should theoretically be easy to prove. However, an area where the correctness of the system may be difficult to ascertain deals with the accuracy of the domain knowledge. If the domain information cannot be exact, the preciseness of the system becomes a debatable quality issue which must be settled based on the desirable characteristics of that particular system. One way to offset the inability to obtain exact domain information is to provide for verifiability through "dynamic evaluation" of the system. The maintainability of an expert system is also a major quality consideration throughout the design phase. The system should demonstrate a capacity to allow for continual updates and maintenance to the dynamically changing knowledge base.

Expert systems are expected to develop and grow beyond the final deployed system. Therefore, adaptation factors such as expandability, flexibility, interoperability, portability and reusability should be researched to determine a system's quality expectations. Flexibility and expandability

must be provided throughout system definition and operation to accomodate the rapid changes in requirements due to refinements, enhancements and/or corrections. The highly iterative nature of prototyping is one convenient tool that can be used as a methodology to support flexibility by encouraging incremental system development. The same prototyping tool can improve the expandability of a system through the provisions of a mechanism to allow for increased software capabilities and/or performance.

Another area where quality is required concerns an expert system's interoperability, which includes the ability to couple with software from another system. Throughout the development effort, attempts should be made to guarantee that the knowledge-based and the conventional software development effort interface properly. In addition, precautions may be necessary to guarantee that the scheduling needs of each software effort complement one another.

Other areas where quality may be necessary concern the portability of the system and the reusability of software for other applications. Frequently, expert systems are developed in a very rich environment. The target environment should be comparable where necessary and vice versa. With respect to reusability, an important work product is the epistemology developed for the domain; it may be reused for other tasks within the same domain.

In summary, quality evaluation is a continuous process whose primary orientation is to build quality into the product. A well defined process model should reflect this idea and recognize that quality attention is something that cannot be left until product delivery. The addressing of performance, design and adaptation issues throughout development will provide the quality evaluation necessary to meet DOD needs.

2.2.6 Contractual Mechanisms

The process of defining a contract for the development of a KBS may require some creative thought. Because the duration/intensity of the system definition phase is difficult to predict, cost estimates may not be very reliable. Specifically, given an initial set of ill-defined requirements, it is difficult to estimate the number of prototypes that will be developed in the process of understanding the problem and postulating a final set of requirements.

Several of the case studies presented in Volume I represent contractual agreements whereby the delivered product was a prototype. In these situations, the contract terminated at the end of the system definition stage. For complicated systems, this approach might suggest the idea of multiple awards for the system definition stage of a project. At the end of this stage, the contractor with the *best* demonstrated prototype and set of proven requirements may be chosen to continue on with system implementation.

The Inference Corporation case study presents another approach. In the contract with American Express, Inference was to build a system that complied with a set of predefined criteria relative to schedule and cost. There was no specific direction in terms of the amount of time or money to be spent during a particular phase. Status was tracked by interim progress reports and prototype demonstrations. Payment was essentially based on a cost basis and, if the end system met the predetermined goals, a balloon profit payment was to be made. This type of contractual strategy provides a profit incentive for the contractor and may enhance the probability of developing a *successful* system.

In the DOD contractual area, there is a definite leaning towards the issuance of fixed priced contracts. In addition, there is also a tendency in some application areas to buy the total system under a single contract (particularly small systems) rather than develop the system in stages (which is usually the case with large systems). In the KBS environment, we have a case where large system procurement policies probably should be followed to buy small systems. The planning for system definition is about all that can be handled when only some vague concept and set of incomplete requirements have been defined. This is typical for the beginning of KBS System Definition.

In the area of contract types, either a Cost Plus or Fixed Price Level-of-Effort is all that could be reasonably used initially. This is based on the inability to predict how many iterations of the System Definition process will be necessary.

Once the system has been defined, it would be reasonable to use a Fixed Price contract. The complete requirements are in hand and a reasonable amount of prototyping completed to accurately plan the implementation and cost of the effort.

2.2.7 Interface to Conventional Software

In the future, DOD systems will include large hybrid systems comprised of both conventional and KBS software. The need for good interfaces between the different types of software will be extremely important for system success. This means that along with the two different types of software communicating with each other, the development teams working on the KBS and conventional software components must also synchronize their efforts.

2.2.7.1 Phasing

In systems containing both conventional and KBS software, there will be a need to coordinate the two development efforts. Certain milestones and reviews of the software will have to coincide. Software development may have to be staggered. While waiting for software from one design approach, a team could potentially work on an unrelated software component. In addition, a limit to the time spent in the system definition phase of KBS software may have to be imposed, so that software integration can be accomplished on schedule.

Certain sections of the system may have to be developed by both design teams. One such section could be the user interface. If the user has to supply information to both portions of the software, then one user interface which obtains both conventional and knowledge base information should be developed. Each discipline must contribute to this effort so that the needs of their software is properly represented and satisfied. A working document that could aid this process is a data flow diagram, which would depict the flow of data between the user interface, controller module (if one exists), and the KBS and conventional software.

The developers of the software will have to communicate and cooperate with each other in order to meet schedules. Ideally, there should be mutual respect between the differing software disciplines. If either of the development teams are uncooperative and difficult to work with, the integration of the software along with the entire system's success may be jeopardized. One common goal to strive for is that of maintaining and developing compatible requirements definitions. Joint meetings wherein

system requirements can be assesed as to their compatibility can facilitate the accomplishment of this goal.

2.2.7.2 Operational Interface

The operational interface between KBS and conventional software is necessary in systems where AI comprises just a portion of the system. In addition, KBS software may need to access databases or conventional software functions. If interfaced inefficiently, the performance of the KBS could be greatly reduced. Recently, some developers of KBS shells have been adding the capability to access existing databases. This demonstrates the importance of the capability. Users of shells typically do not want to duplicate information already stored in a database.

In hybrid systems, the standards of how data will be passed and controlled between software components should be agreed upon early in the life cycle. If accomplished, both types of developers can prepare for the interfacing and not have any "surprises" come integration time. As a consequence, time will have to be set aside for the development teams to hash out the control and data issues.

Another operational issue involves the electronic interface between different computers. If the system as a whole runs on more than one computer, then electronic interfacing is a critical factor. In this situation the details of how the different processors will communicate must be addressed and resolved to everyone's satisfaction.

2.2.8 Interface to Systems Engineering

In large system developments, systems engineering requirements are typically invoked contractually by specifying MIL-STD-499 as a compliance document within a contract. This standard, although it has not been updated since May 1974, continues to provide the system engineering precepts that are important to an orderly system development process.

2.2.8.1 System Design Activities

The design activities commence with the conduct of mission analyses, progressing to system functionality definition, and finally ending with the allocation of functions and subfunctions to hardware and software components that make up the system.

Mission analysis output is the verification of existing system requirements or the development of new requirements for the system. System operational characteristics, *mission objectives*, *threat*, *environmental* factors, functional requirements, technical performance and other factors are examined to properly characterize the system prior to large scale design and development commitment. The output of mission analysis may well provide a precise indication that KBS techniques will need to be applied. These mission related characteristics provide the foundation and system characterization from which to begin functional analysis.

Functional analysis includes analyzing and defining system functions and sub-functions, and identifying design alternatives for meeting mission needs. Trade-off and optimization studies are conducted to ensure any resulting system meets mission requirements in a cost-effective manner. During this process, the KBS application may be further defined and refined such that the overall

system design clearly identifies the division of labor between the KBS and conventional software components.

The allocation of functions and sub-functions to hardware and software components represents the final step in the system design process wherein discrete components are identified for further design and development. This process is the subject of further definition and trade-off studies to ensure that mission requirements can still be achieved in a cost-effective manner. At this juncture of the development process, there should be a clear definition of where KBS are appropriate and the division of the system into its component parts should reflect this allocation of functions. There should be distinctions between the various elements of hardware and software to include recognition of KBS and conventional software application. This division provides the starting point wherein a KBS model can begin to be applied.

2.2.8.2 System Interfaces

Two critical aspects of interface definition and control are intrasystem and intersystem compatability requirements. The system engineering process must not only ensure compatible interface design but must also be concerned with interface implementation to make sure that what is built will interface properly with appropriate components. There is a need to have a continuous interface engineering presence on all programs to insure that hardware works with software, software interfaces with software, and that systems/subsystems interface with other systems/subsystems. It is critical that KBS applications be a participant in this interface definition and implementation process. If a KBS is to interface with some element of conventional software from a data or control standpoint, this interface needs to be defined and then controlled throughout the development process, thus providing assurances that all components will work when integrated and tested. The KBS model developed must reflect, and not prohibit, this continuous attention to interfaces and provide control at the same time providing flexibility for controlled changes.

SECTION 3

Derived KBS Models

3.1 Initial Model Based on KBS Development Characteristics

Figure 3-1 presents a high level model, showing the relationship between the three major processes of KBS software development. Iterative development is a common aspect of each process, although it is most highly emphasized within the system definition process. Although implied by Figure 3-1, the duration of each process is not necessarily the same for every application. For example, a very complex project or one with very uncertain requirements may require a considerable system definition effort and a relatively small system implementation effort. Other variations are also possible.

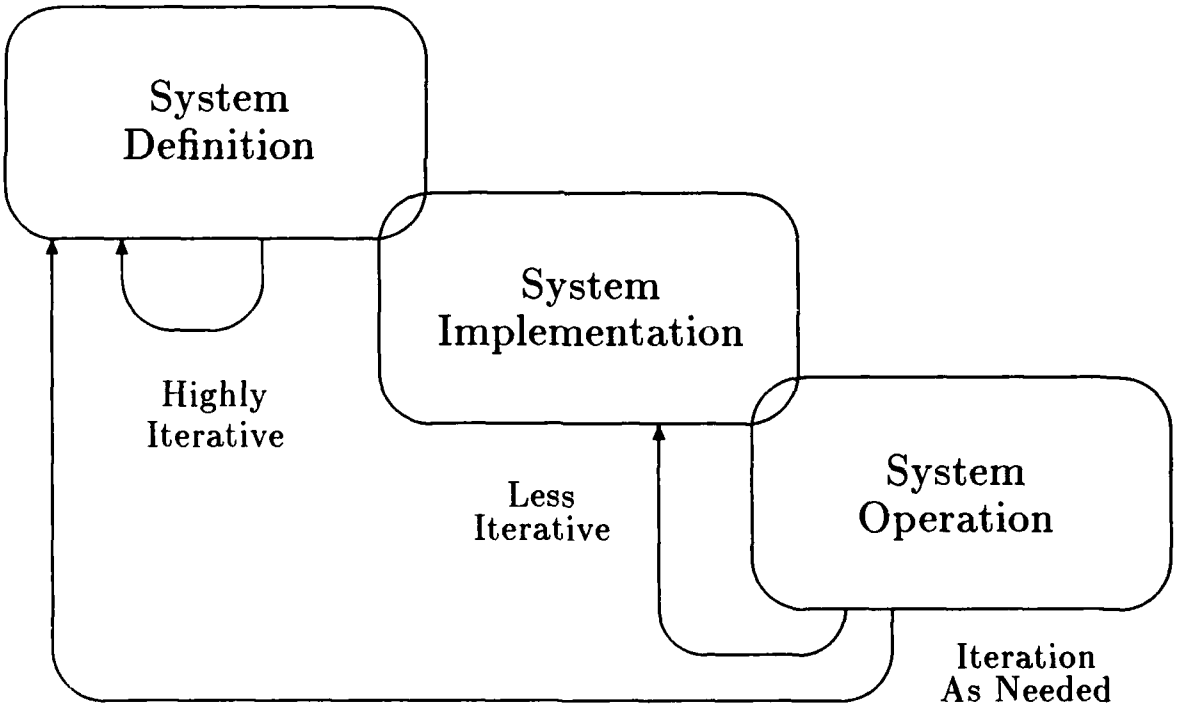


Figure 3-1: Top Level KBS Process Model

Figure 3-2 provides a more detailed presentation of the KBS process model, noting the significant activities or subprocesses that occur within each major process. A detailed description of the model is presented below.

3.1 *Initial Model Based on KBS Development Characteristics*

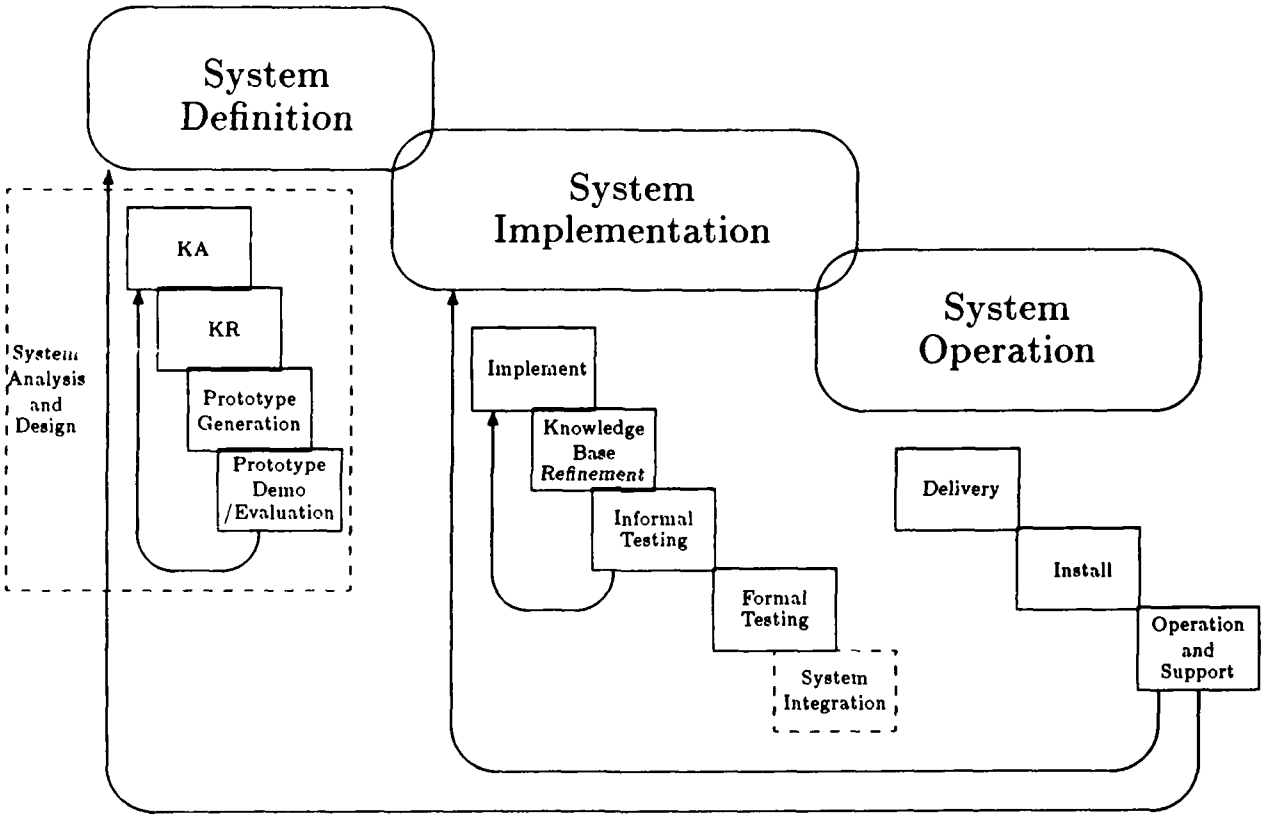


Figure 3-2: Detailed KBS Process Model

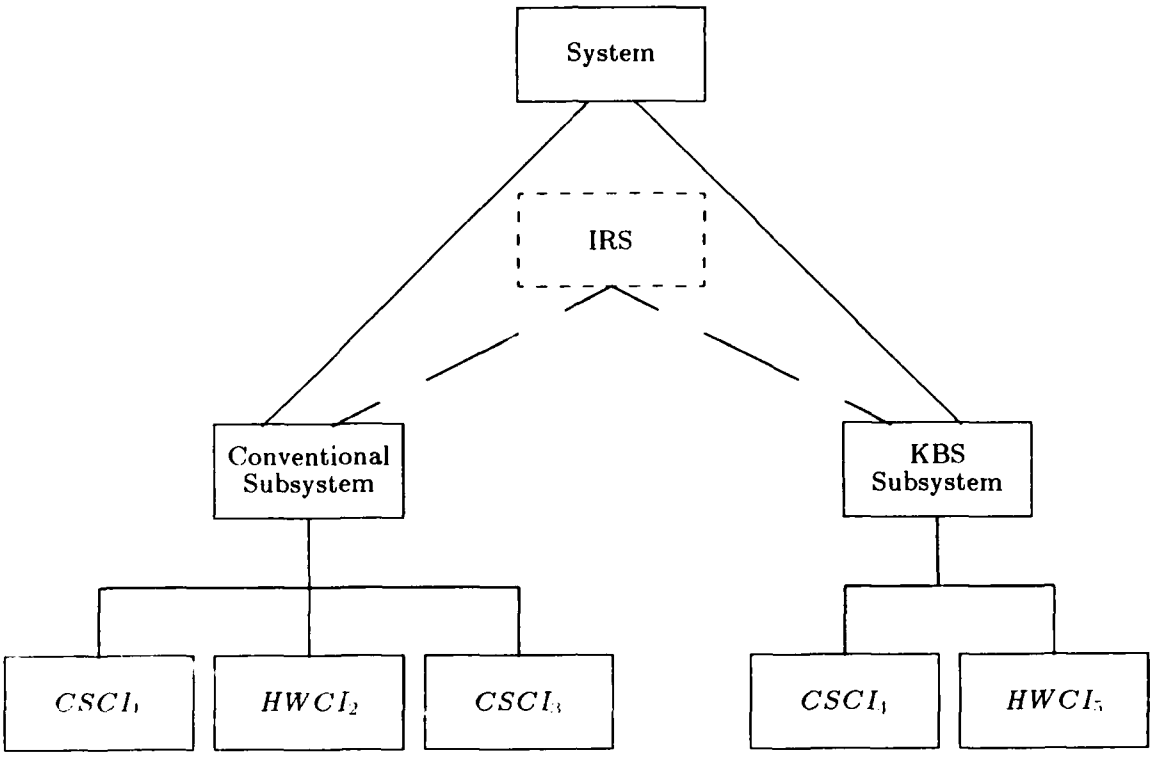


Figure 3-3: Sample Hybrid System Organization

3.1.1 System Definition

The System Analysis and Design activity noted within the dashed box in Figure 3-2 represents the system engineering process that occurs as part of defining a system architecture to satisfy DOD operational needs. The design activity consists of conducting analyses and trade offs between hardware and software approaches in an attempt to optimize a cost effective system design approach. The primary output from the system engineering process is a System Specification that defines the system level performance and functional requirements as well as component hardware and software requirements. When AI techniques are to be investigated and applied within a system, the System Specification would define the high level requirements for the KBS segment and include these documented requirements in paragraph 3.4 of the System Specification (see Data Item Description DI-CMAN-80008A (draft), paragraph 10.2.5.4). These requirements would then be further explored and refined via the KBS process model in Figure 3-2 and documented as a part of a KBS Segment Specification. Figure 3-3 provides a sample hybrid system organization illustrating both conventional and KBS CSCI's and Hardware Configuration Items (HWCI). In those cases where the KBS is a stand alone system and not part of a larger system, the KBS development process would ignore the System Analysis and Design activities and the preparation of a System Specification and begin the definition process by defining a KBS Segment Specification.

The primary goal of the KBS definition process is to understand the problem at hand in order

to establish a complete/accurate set of KBS requirements and design a system to satisfy these requirements. The process is focused on the use of exploratory programming as a technique for assessing project feasibility and exploring potential solution strategies. Specific activities that comprise the system definition process include:

- Knowledge Acquisition
- Knowledge Representation
- Prototype Generation
- Prototype Demonstration/Evaluation

As these activities are revisited, the problem becomes more clearly defined, and the software requirements begin to emerge. Initially the requirements are dynamically characterized and subject to change as the problem is better understood. At some point during the iteration process, the requirements as well as the solution strategy begin to stabilize. Namely, as the requirements are evolving, ideas are formulated which address *how* the system can best meet the established requirements. Ideally the user community should be involved in the activity of prototype demonstration/evaluation. User involvement in this activity greatly enhances the likelihood of developing requirements which adequately address the problem at hand. Once the customer and the developers are satisfied that the requirements accurately define the right system, and that the prototypes have suggested an appropriate system design approach, the system definition process is complete.

In some cases, the culmination of the system definition process may reveal that the system is best implemented using conventional techniques. Prior to prototyping, however, the solution was not apparent. If this is the case, development of the system can be managed using a conventional model such as DOD-STD-2167.

Furthermore, the system definition process may be an end in itself. For example, a customer may only be interested in a feasibility study for a particular problem. In another situation, a customer may not want to commit to implementing a system until there is some indication of the level of effort involved. At the end of the system definition process, the level of effort required to implement the system can be more accurately defined since the requirements are better understood.

3.1.2 System Implementation

As indicated in Figure 3-2, the system implementation process follows the definition process, assuming the goal is to build a full scale development system. The key activities that comprise this second major process are:

- Implementation
- Knowledge base refinement
- Informal testing
- Formal testing

- System integration

During this process the system design is implemented in the target environment, which may be different from the prototyping environment. If this is the case, the implementation process may be significantly complicated by the need to consider new hardware and, perhaps, a different programming language. In any case, the implemented system must *completely* address the problem at hand. The system as implemented will certainly include more functionality than the prototypes generated during system definition. Namely, the knowledge base will likely be refined or augmented to cover more cases than the prototype. In the process, the knowledge base becomes more robust, perhaps able to deal with uncertainty more so than the prototype.

The implemented system must also be equipped with error handling facilities, which generally are not of great concern in the system definition process. As implemented, there may be a need to consider program efficiency in terms of memory utilization and run time. Efficiency optimization may involve modification of the knowledge base, inferencing mechanism and/or data structure representation.

In general, it may be better to rewrite the entire system during the implementation process as opposed to evolving the latest prototype. If the target environment differs from the prototyping environment, rewriting the system will be a necessity.

As the system is implemented, informal testing occurs to assess the adequacy of the progressing product. Identified deficiencies must be corrected by revisiting implementation of the effected code segments.

Near the end of the system implementation process, formal acceptance testing of the KBS software is performed for customer acceptance purposes. In hybrid systems containing both KBS and conventional software components, system integration is a necessary activity to bring all system components together into an integrated environment and to test the system as an integrated whole. The dashed box in Figure 3-2 indicates that system integration occurs only when there are multiple KBS and conventional components to be integrated. Namely, a standalone KBS which does not interface with any conventional software does not require integration. System implementation is considered complete when the system is accepted by the customer.

3.1.3 System Operation

The system operation process involves the following activities:

- Delivery
- Installation
- Operation and Support

Support may involve correcting deficiencies or enhancing the system. Major changes to the system may call for development to cycle back to the system definition process to prototype the enhancements. Minor changes will likely cycle development back to the system implementation process. System operation is simply a repeat of selected portions of earlier processes.

3.2 Postulated Model Encompassing DOD Needs

Figure 3-4 presents a recommended model for guiding the development of knowledge based systems. The model maintains the set of activities identified in the previous section and adds a set of products, reviews and baselines. The additional elements shown in Figure 3-4 were primarily selected to facilitate management of the developing system and ease operation and support of the delivered product. At first glance, the model appears to be very similar to DOD-STD-2167 and the released Draft of DOD-STD-2167A dated 1 April 1987. The reviews and baselines are essentially equivalent to their DOD-STD-2167/2167A counterparts. It is primarily the products that differ, and hence the interpretation of *what* is to be reviewed and baselined.

The following subsections present a more detailed discussion of the model using the DOD-STD-2167A product descriptions (Data Item Descriptions) as the point of departure for the KBS products. This approach was chosen because the rewritten Data Item Descriptions under the Draft 2167A represent a more acceptable documentation set for KBS comparisons.

3.2.1 System Definition

System definition is a highly iterative activity wherein multiple prototypes are built in the process of understanding the problem at hand and developing a complete set of requirements. Despite the uncertainties associated with a proposed system, the concepts of deliverable products, customer attended reviews, and baseline configuration mechanisms provide a means for tracking system progress. The following subsections fully describe the System Definition portion of the interface model presented in Figure 3-4.

3.2.1.1 Products

Within the KBS definition process, the recommended products are intended to provide customer visibility and control over this highly iterative process. In addition to the prototype itself, the products as a whole capture the underlying philosophy of the selected development approach and provide meaningful information which can be referenced long after the system becomes operational. All documents shown in Figure 3-4 satisfy both customer and developer needs; none are intended to interfere with or hamper the development process. The specific documents required during System Definition are proposed as follows:

- Preliminary Interface Requirements Specification;
- Preliminary KBS Segment Specification;
- Software Development Plan A;
- Interface Requirements Specification;
- KBS Segment Specification;
- Preliminary Functional Design Document; and

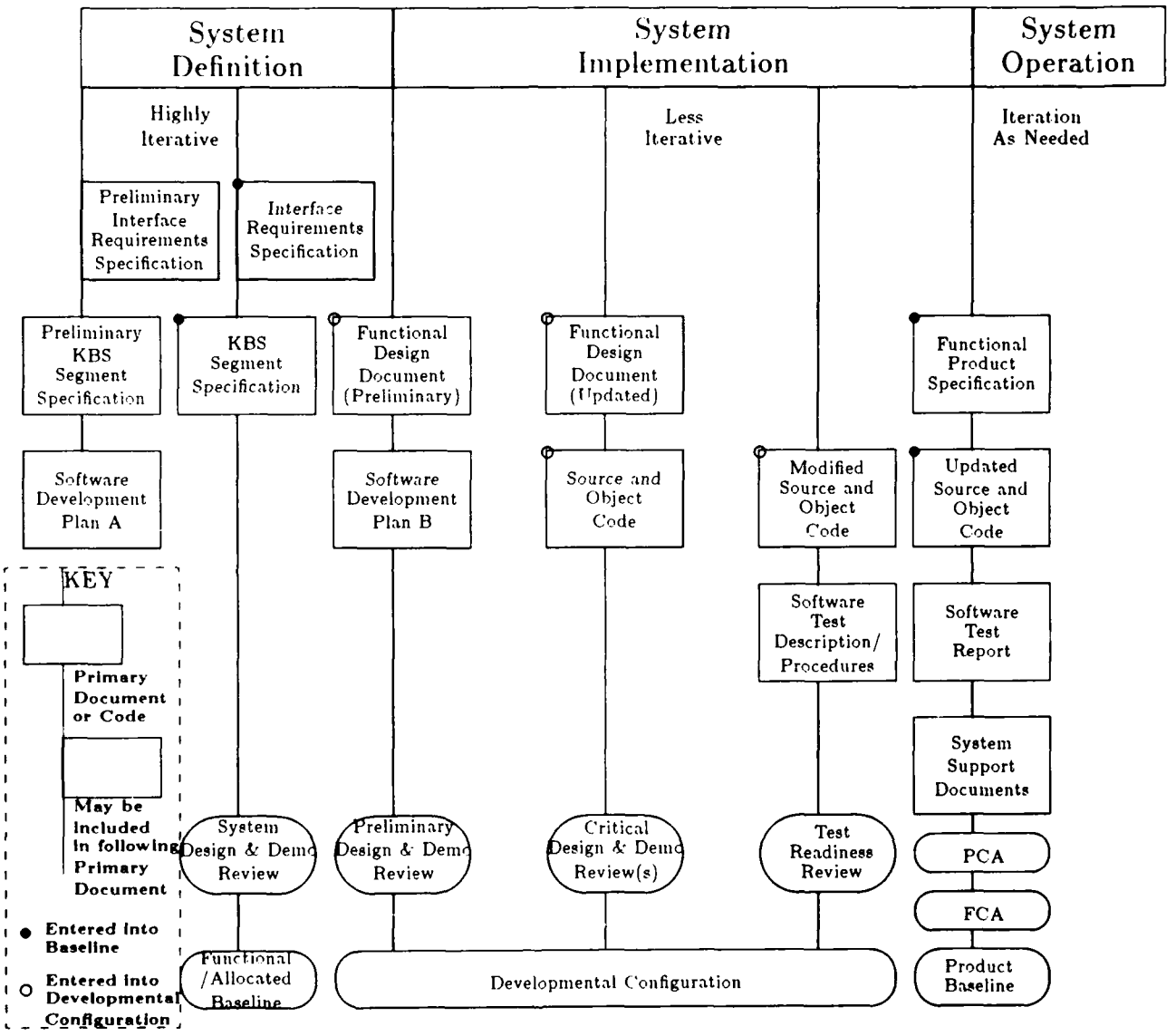


Figure 3-4: KBS Developer/Customer Interface Model

3.2.1 System Definition

- Software Development Plan B.

Each of these seven documents is described in the following paragraphs.

3.2.1.1.1 Preliminary Interface Requirements Specification (PIRS) The purpose of the PIRS is to provide system interface information on the CSCI, HWCI and critical item level. If one considers the KBS software itself as one CSCI, then information must be provided relating the KBS CSCI to other system components. In a hybrid system, external interfaces to conventional software CSCI's must be defined. In a cooperative KBS scenario, the manner in which one KBS communicates with another must also be defined. User interface details should also be included in the PIRS. Internal KBS interface information should be documented, if applicable.

Figure 3–5 presents a high level outline of the DI-MCCR-80026A (draft) data item description for an IRS under DOD-STD-2167A (draft). The PIRS for a KBS or hybrid system consisting of both knowledge-based and conventional software could follow this outline reasonably well.

Because of the possibility that the specific partitioning of the system is not always well defined at the beginning of a program, many of the paragraphs for the PIRS may contain the notation "*to be determined*".

Depending on the specific program at hand, it may be desirable to eliminate the PIRS as a separate document. Instead, the salient interface information can be included as part of the Preliminary KBS Segment Specification discussed below.

1. Scope
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview

2. Applicable Documents

3. Interface Requirements
   - 3.1 Interface Relationships
   - 3.2 CSCI-to-CSCI Interface Requirements
   - 3.3 CSCI-to-HWCI or Critical Item Interface Relationships
   - 3.4 Interface Documentation

4. Interface Qualification Requirements
6. Notes

Appendixes

Figure 3–5: Interface Requirements Specification Outline

3.2.1.1.2 Preliminary KBS Segment Specification (PKSS) The purpose of the PKSS is to provide a high level description of the proposed system, its envisioned segments and associated requirements. Because the system itself is not likely to be well understood at the outset, the initially partitioned system and stated requirements may only represent a *best guess.* Figure 3–6 presents a suggested outline for the final KBS Segment Specification which can be followed in developing the PKSS. At the initiation of system definition, many of the paragraphs for the PKSS will contain scant information or even notations of *"to be delivered".* Nonetheless, with the information available, the PIRS and PKSS present a starting point from which the system can begin to be defined using exploratory programming and prototyping techniques.

A more detailed discussion of Figure 3–6 is presented in the KBS Segment Specification section below.

3.2.1.1.3 Software Development Plan A (SDP-A) Software Development Plan A presents management's strategy for reaching the following major goals of system definition :

- Generate a complete and accurate set of software requirements;
- Complete the bulk of the knowledge acquisition effort;
- Establish the method(s) of knowledge representation; and
- Develop design ideas to facilitate the system implementation process.

Because the outcome of system definition is unknown at this point, it is wise to postpone the development of plans for managing the system implementation effort. Software Development Plan B, discussed later in this section, covers the management and planning aspects of System Implementation.

The content and format of SDP-A can follow that of DI-MCCR-80030 (draft) under DOD-STD-2167A (draft) reasonably well. A top level outline for this particular data item is shown in Figure 3–7. Note, however, that several of the subparagraphs under paragraph 4.2 of DI-MCCR-80030 (draft) entitled "Software Standards and Procedures" will need to deviate from the focus on activities of :

- Software Requirements Analysis;
- Preliminary Design;
- Detailed Design;
- Coding and Unit Testing;
- CSC Integration and Testing; and
- CSCI Testing.

Instead, many of the subparagraphs under paragraph 4.2 should be restructured to properly reflect the activities that do occur during system definition. Specifically, the major activities listed below are performed iteratively in the process of defining and refining system software requirements :

3.2.1 System Definition

1. Scope
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview
   - 1.4 System Definition
     - 1.4.1 Missions
     - 1.4.2 Threat

2. Referenced Documents

3. System Requirements
   - 3.1 System Domain
     - 3.1.1 Knowledge Base Coverage
     - 3.1.2 Inferencing Mechanism
   - 3.2 System Modes and States
   - 3.3 System Capabilities
     - 3.3.X System Capability Name and Number
   - 3.4 System Capability Relationships
   - 3.5 *Segment Allocation*
   - 3.6 Configuration Allocation
     - 3.6.X HWCI or CSCI Name and Number (KBS software = a CSCI)
   - 3.7 Detailed Functional and Performance Requirements
     - 3.7.1 HWCI
     - 3.7.2 CSCI (KBS software = a CSCI)
       - 3.7.2.1 Inputs
       - 3.7.2.1 Processing
       - 3.7.2.1 Outputs
   - 3.8 Government Furnished Property Usage Requirements

   ⋮

   Continue with 2167A DI-CMAN-80008A (draft)

   ⋮

   - 3.25 Precedence
4. Qualification Requirements
5. Preparation for Delivery
6. Notes

Appendixes

Figure 3-6: KBS Segment Specification Outline

1. Scope
   - 1.1 Identification
   - 1.2 Purpose
   - 1.3 Introduction
   - 1.4 Relationship to Other Plans

2. Referenced Documents

3. Software Development Management
   - 3.1 Project Resources and Organization
   - 3.2 Schedule and Milestones
   - 3.3 Risk Management

   ⋮

   Continue with 2167A DI-MCCR-80030 (draft)

   ⋮

   - 3.11 Problem/Change Report

4. Software Engineering
   - 4.1 Resources and Organization - Software Engineering
   - 4.2 Software Standards and Procedures
   - 4.3 Engineering Research Activites
   - 4.4 Non-developmental Software

5. Formal Software Testing
6. Software Product Evaluations
7. Software Configuration Management
8. Other Software Development Functions
9. Notes

Appendixes

Figure 3-7: Software Development Plan A Outline

3.2.1 System Definition

- Exploratory programming;
- Knowledge acquisition;
- Knowledge representation;
- Prototype generation; and
- Prototype evaluation.

During the discussion on knowledge acquisition, information pertaining to the selection of a domain expert(s) should be included. The accessibility and availability of the domain expert(s) should also be stated.

3.2.1.1.4 Interface Requirements Specification (IRS) As shown in Figure 3-4, the IRS and KBS Segment Specification are scheduled for delivery sometime well into the System Definition process. By this time, multiple prototypes will have been developed and the partitioning of the system into components should be better defined. As such, the interfaces between the components can be established and documented in the IRS. This documentation effort will essentially consist of updating the PIRS according to the outline shown in Figure 3-5.

Depending on the specific program, it may be preferable to eliminate the IRS as a separate document. Instead, the interface data can be included as part of the KBS Segment Specification discussed below.

3.2.1.1.5 KBS Segment Specification (KSS) The KSS provides a description of the proposed system on an overall as well as a CSCI/HWCI level. The documentation effort involves updating the PKSS according to the outline shown in Figure 3-6. Notice that the outline shown basically follows the format of data item DI-CMAN-80008A (draft) under DOD-STD-2167A (draft). The differences lie in Sections 3.1 and 3.7 shown in Figure 3-6. In Section 3.1 of Figure 3-6, the contractor must describe requirements associated with the system domain. Specific elements include knowledge base coverage and the inferencing mechanism. Namely, what aspects of the overall problem will be addressed by the knowledge base itself? In addition, from the prototyping that has been done, what is the recommended strategy in terms of inference processes?

Section 3.7 of Figure 3-6 presents functional and performance requirements on the CSCI/HWCI level. Rather than place this material in a separate document, such as the Software Requirements Specification under DOD-STD-2167A (draft), we recommend inclusion in the KSS. One should consider the KBS portion of a system as a single CSCI for which software requirements would be stated in terms of inputs, processing, and outputs. Subsections under Section 3.7 should be included for each KBS that comprises the total system.

3.2.1.1.6 Preliminary Functional Design Document (FDD) At the end of System Definition, multiple prototypes have been built and many ideas have been generated in terms of the system design to be implemented. These ideas along with the rationale behind the approaches selected should be captured in the preliminary Functional Design Document (FDD). A suggested outline for the preliminary FDD is shown in Figure 3-8.

The preliminary FDD presents a conceptual design of the system in terms of the target environment. The partitioning of the system into specified components should be documented and substantiated. The preliminary FDD also contains a section describing the knowledge base. Near the end of System Definition, the knowledge engineers usually have developed strong opinions concerning knowledge representation. The conceptual design of the knowledge base as well as the justification behind the design should be captured at this point. Furthermore, if the knowledge base has been partitioned into subsets of knowledge with differing methods of syntax and taxonomies employed, a thorough description should be provided for each subset.

Design information concerning the inferencing mechanism chosen is also contained within the preliminary FDD. If the knowledge base has been partitioned, a description of the inference process used to navigate through the subsets of knowledge should be provided.

Conventional software components should be defined and substantiated (i.e. why did the contractor choose conventional techniques over KBS techniques for this particular system function?) Interface descriptions amongst the system components should also be provided. In addition, the use of off-the-shelf software should be specifically identified as part of the design solution.

Given the information it contains, the preliminary FDD can be thought of as a high level design document from which System Implementation can begin.

3.2.1.1.7 Software Development Plan B (SDP-B) Once the System Definition process is complete, requirements and design ideas pertinent to the final system should be well established. Not until this time is the contractor able to plan for the System Implementation process. In documenting the management strategy for System Implementation, the outline shown in Figure 3-7 can be followed.

SDP-B essentially represents a new software development plan geared towards the System Implementation effort as opposed to the System Definition process. Although System Implementation is much less iterative than System Definition, the activities still deviate from the standard waterfall approach. The concentration is centered on functional designs and demonstrations of the system in the target environment throughout the implementation period. Consequently, the subparagraphs under paragraph 4.2 of SDP-B should be structured to reflect these activities.

In addition, under paragraph 5 entitled "Formal Software Testing", one should include the *class* of test cases to be covered during formal system evaluation activities. Specific cases that address the *class* of test cases will be included in the Software Test Description/Procedures document discussed in Section 3.2.2.1.

3.2.1 System Definition

1. Scope
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview

2. Referenced Documents

3. System Components
   - 3.1 Overview
   - 3.2 Rationale

4. Knowledge Base
   - 4.1 Knowledge Representation
     - 4.1.1 Syntax
     - 4.1.2 Taxonomies
   - 4.2 Rationale

5. Inference Processes and Mechanisms
   - 5.1 Reasoning Method
   - 5.2 Control Structures
   - 5.3 Relation to Other Processes
   - 5.4 Rationale

6. User Interface

7. Interfaces
   - 7.1 Internal Interface
   - 7.2 External Interface

8. Notes

Appendixes

Figure 3-8: Functional Design Document Outline

3.2.1.2 Reviews

The System Design and Demonstration Review (SD&DR) is the formal review mechanism scheduled for the KBS during System Definition. The purpose of this customer/user attended review(s) is to evaluate the KBS Segment Specification and the current state of the most recent prototype. Consequently, the partitioning of the system and its associated requirements will be scrutinized for accuracy, completeness, correlation, optimization and risk. The SD&DR will be conducted when the System Definition effort has proceeded to the point where system characteristics are defined, configuration items identified and requirements established. Depending on system complexity, more than one SD&DR may be scheduled during System Definition.

Note that the preliminary FDD and SDP-B are reviewed at the beginning of the system implementation process (see section 3.2.2.2).

3.2.1.3 Baselines

When the IRS and KSS have been accepted by the customer in terms of completely defining the system's requirements, these documents shall be entered into the Functional/Allocated baseline. In this manner, there is a level of formal control over the finalized set of requirements and system partitioning. Changes to the Functional/Allocated baseline requires an Engineering Change Proposal (ECP) as defined in MIL-STD-480.

3.2.2 System Implementation

System Implementation is the process of developing the final product for the target environment. Knowledge base refinement and informal testing of the incrementally built system occurs in an iterative fashion until the system is ready for formal test procedures. The following subsections describe the products, reviews and baselines depicted in Figure 3–4 under System Implementation.

3.2.2.1 Products

Within the System Implementation process, the recommended deliverable products consist of documentation and code. Delivery of the code occurs at the end of this process when the system has been tested and accepted by the customer. The specific documents called for consist of the following:

- Functional Design Document (Updated);
- Software Test Description/Procedures Document;
- Functional Product Specification;
- Software Test Report; and
- System Support Documents.

Each document is described in the following paragraphs.

3.2.2 System Implementation

3.2.2.1.1 Functional Design Document (Updated FDD) As System Implementation begins, the contractor starts to develop software for the target environment based on the design concepts presented in the preliminary FDD. Because implementation may reveal some flaws in the initial design work or the need for additional knowledge acquisition/representation, some prototyping will occur during System Implementation. As the design ideas in the preliminary FDD are updated and/or modified, they need to be captured in written form. At some point during the System Implementation process, the preliminary FDD should be updated. The updated FDD will follow the same outline as the preliminary FDD shown in Figure 3–8.

Depending on the size and complexity of the system at hand, more than one updated version of the FDD may be required.

3.2.2.1.2 Software Test Description/Procedures Document (STD) The STD document identifies the information necessary to conduct formal CSCI testing for the KBS. The format of the document can follow that of DI-MCCR-80015A (draft) reasonably well as outlined in Figure 3–9.

Within paragragh 5 of DI-MCCR-80015A (draft), the procedures to be used to perform static evaluation of the knowledge base should be included in addition to the specific test cases described relative to dynamic evaluation of the KBS.

3.2.2.1.3 Functional Product Specification (FPS) The FPS consists of the design documents as well as software listings that pertain to the implemented KBS. A suggested outline for the FPS is presented in Figure 3–10.

Note that in subparagraph 3.1.1, the Functional Design data may either contain or reference the appendix that contains the most recently updated FDD.

3.2.2.1.4 Software Test Report (STR) The STR contains a record of the formal testing performed for the KBS CSCI. The format of the STR can follow that of DI-MCCR-80017A (draft) under DOD-STD-2167A (draft). A high level outline of this data item description is shown in Figure 3–11.

Note that in paragraphs 3 and 4, the sections are broken down by individual tests.

3.2.2.1.5 System Support Documents The requirement for the types of different System Support documents should be determined by the customer on a system-by-system basis. Specific documents that may be required include :

- Software User's Manual;
- System Operation Manual;

1. Scope
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview

2. Referenced Documents

3. Formal Test Identification
   - 3.1 General Test Requirements
   - 3.2 Formal Test Classes
   - 3.3 Formal Test Levels
   - 3.4 Formal Test Definitions
   - 3.5 Formal Test Schedule
   - 3.6 Data Recording, Reduction and Analysis
   - 3.7 Formal Test Reports
   - 3.8 Assumptions and Constraints

4. Formal Test Preparations

5. Formal Test Descriptions

6. Notes

Appendixes

Figure 3-9: Software Test Description/Procedures Outline

3.2.2 System Implementation

1. Scope
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview

2. Referenced Documents

3. Requirements
   - 3.1 CSCI Listings
     - 3.1.1 Functional Design
     - 3.1.2 Knowledge Base Listings
     - 3.1.3 Rule Relationships
     - 3.1.4 KBS Shell Listing (if available)
     - 3.1.5 Inference Mechanism Listing
     - 3.1.6 User Interface Listing (if available)
     - 3.1.7 Other Related Listings

4. Notes

Appendixes

Figure 3-10: Functional Product Specification Outline

1. Scope
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview

2. Referenced Documents

3. Test Overview

4. Test Results

5. CSCI Evaluation and Recommendations

6. Notes

Appendixes

Figure 3-11: Software Test Report Outline

- Firmware Support Manual;
- Computer Resources Integrated Support Document; and
- Version Description Document.

The format of these documents can follow the respective data item descriptions under DOD-STD-2167A (draft).

3.2.2.2 Reviews

The Preliminary Design & Demonstration Review (PD & DR) kicks off the System Implementation process. Final documents from System Definition, the preliminary FDD and SDP-B, are evaluated and the final prototype is demonstrated. Review of these products should concentrate on the proposed system partitioning, knowledge base coverage, knowledge representation including a walk-through of the "rules", and inferencing strategies relative to the target environment.

The Critical Design & Demonstration Review (CD & DR) should be scheduled sometime into the System Implementation process. Depending on the system schedule and complexity, multiple CD & DR's may be planned. The purpose of the CD & DR is to review the current status of the system. This includes a review of the updated FDD and a demonstration of the system as implemented to date. The review should ensure that the knowledge base completely covers the predefined domain and that the system is making progress towards satisfying the baselined requirements.

The Test Readiness Review (TRR) is held near the end of System Implementation to determine if the test procedures are complete and if the KBS is ready for formal CSCI testing. The STD document is reviewed and the test procedures compared with the test planning documented in SDP-B. The customer may also review the results of the informal testing efforts.

At the end of System Implementation, Physical and Functional Configuration Audits are held. The Physical Configuration Audit (PCA) is a technical evaluation of the KBS to verify that the "as-built" CSCI agrees with the technical documentation that describes it. The Functional Configuration Audit (FCA) is performed to validate that the KBS has been completed satisfactorily and achieves the performance and functional characteristics as specified in the Functional/Allocated baseline.

3.2.2.3 Baselines

Two baselines are established during the System Implementation process: the Developmental Configuration and the Product baseline. The Developmental Configuration is a contractor controlled mechanism which is used to track the FDD's and the source/object code as the system progresses towards completion. The precise tracking mechanisms and controls are defined in SDP-B.

The Product baseline is a formal or customer controlled baseline into which the FPS and the final versions of the source and object code are entered. Changes to the Product baseline require an ECP.

3.2.3 System Operation

The System Operation process involves support on an *as needed* basis. As enhancements are requested, activities may cycle back to System Definition or Implementation as a function of the degree of system modification. In any case, changes made must be reflected in associated documentation which should be entered into the existing baselines along with the updated source and object code.

3.3 Advantages of the Postulated Model(s)

3.3.1 Resolution of Common Software Problems

Historically, the issues of control, visibility, quality, and supportability have been common problems in the software life cycle. The following paragraphs discuss how the postulated KBS model attempts to resolve these issues.

Control of the developing software is addressed by the inclusion of baselines and the developmental configuration into the KBS development process. The baselines provide a means of formally tracking and controlling the evolving software. The developmental configuration provides a similiar informal mechanism wherein the contractor is responsible for defining and maintaining internal controls.

The documents, reviews and demonstrations called for by the KBS interface model provide visibility into the development process. They also provide a means by which the quality of the software can be assessed. The iterative design approach allows for early and constant verification/validation of the quality of the system. As a result, flaws in the requirements for the system can be found early in the development process thus avoiding costly design deficiencies when found late in the software life cycle. The model also allows for audits to be performed in which the development methodologies used can be compared to those stated in the development plans.

The documents produced by applying the KBS model contain the necessary information to allow for the support of the system. In addition, the justification behind the early design decisions included in the KBS documents provides useful information that enhances the supportability of the system. The cycling back to the development process that occurs when the system undergoes design changes and enhancements is provided for in the postulated KBS model. Reiteration will provide a system of updated documentation as changes are made.

3.3.2 Meets DOD Management Needs

In terms of management needs, the KBS model requires enough products to give the managers visibility and control over the development process. The delivery of the SDP-B prior to the start of System Implementation provides a more realistic scoping of the work to be managed and controlled during that stage of the process. This approach recognizes that the complete development process can not be detailed at the start of System Definition and that replanning is a necessary part of developing KBSs. This planning approach provides evolving visibility and control specifics to be addressed as the KBS evolves.

With the inclusion of prototyping in the KBS model, the engineering process can work effectively. The iterative nature of the model should help the engineers build quality systems. The model does not over burden the engineers with a large amount of documentation. The documents required are those that are needed in order to support the system.

As previously stated, verification/validation of the system begins early on and continues throughout the development process. The testing approach presented in section 2.2.4 will provide an adequate validation of the system in terms of KSS requirements satisfaction. Namely, the testing approach will demonstrate the performance, functions and interfaces of the system with the goal of proving that the developed system is of sound quality and works as expected.

3.4 Comparison of KBS and 2167 Interface Models

3.4.1 Overview

The KBS interface model was designed to be compatible with both DOD-STD-2167 and the 2167A (draft) interface model. This was done to accomodate the development of hybrid systems containing both conventional and KBS software. Figure 3-3 depicts how a hybrid system may be organized. As indicated, one CSCI of the system would be comprised of the KBS. In systems with more than one KBS, each knowledge base portion would be one CSCI. The KBS should not be broken down into several CSCIs, as this would prove to be a difficult task and one that would most likely hinder the success of the system.

Regardless of the system type, KBS or hybrid, the system analysis and design activity and its products are largely unchanged; except for the identification of what parts will be conventional software and what parts will be implemented using a KBS approach. Also, the system support activities and products remain the same. Table 3.4.1-1 depicts the 2167/2167A (draft) phases and how they correlate to the KBS processes. As indicated, the phases of pre-software development, software requirements analysis, and preliminary design, or their equivalent, are performed during the KBS system definition process. Likewise, detailed design, coding and unit testing, CSC integration and testing, CSCI testing, and systems integration and testing occur during the KBS system implementation process. Lastly, the phase of production and deployment take place during the KBS system operation process.

3.4.2 Products

Table 3.4.2-2 depicts the correlation between 2167A (draft) and KBS products. Some of the products are the same in both models and therefore do not appear in this table. Namely, individual support documents such as Software Users Manual, Computer System Operations Manual, Firmware Support Manual, Computer Resources Integrated Support Document, and Version Description Document do not appear. Integration documents such as System Integration Test Plans, System Integration Test Procedures, and System Integration Test Report are also unchanged. The Operational Concept Document is another product which is the same in both models.

The PIRS and IRS contain the same information in both models. In the KBS model, we suggest including these documents in the PKSS and KSS respectively. The PKSS and KSS documents

3.4.2 Products

Table 3.4.1-1: Mapping of Phases to Processes

2167/2167A (Draft)	KBS
Pre-Software Development: System Concepts System Requirements Analysis Software Requirements Analysis Preliminary Design	System Definition
Detailed Design Coding and Unit Testing CSC Integration and Testing CSCI Testing Systems Integration and Testing	System Implementation
Production and Deployment	System Operation

Table 3.4.2-2: Mapping of Products

2167A (Draft)	KBS
Preliminary Interface Requirements Specification	Preliminary Interface Requirements Specification
Preliminary System Segment Specification Preliminary Software Requirements Specification	Preliminary KBS Segment Specification
Interface Requirements Specification	Interface Requirements Specification
System Segment Specification Software Requirements Specification	KBS Segment Specification
Software Development Plan	Software Development Plan A Software Development Plan B
Software Top Level Design Document	Functional Design Document (Preliminary)
Software Detailed Design Document	Functional Design Document (Updated)
Software Test Description (Test IDs) Software Test Description (Cases) Software Test Description (Procedures)	Software Test Description/Procedures
Software Test Reports	Software Test Reports
Software Product Specification	Functional Product Specification
System Support Documents	System Support Documents

contain added information when compared to the 2167A (draft) equivalent products. The additional information includes: system domain information, inferencing mechanism to be used, and software requirements information. The software requirements information is extracted from the 2167A (draft) Preliminary Software Requirements Specification (PSRS) and Software Requirements Specification (SRS) documents. Because of the small development team concept typically employed, treating requirements at the segment level for the software readily lends itself to most KBS development approaches.

The 2167A (draft) SDP document is divided into two documents in the KBS model: SDP-A and SDP-B. The SDP-A contains information pertinent to the software development activities during the System Definition process and the SDP-B contains information pertinent to the software development activities during the System Implementation process. As the activties of KBS software development are different from those of conventional software development, the section of "Software Standards and Procedures" will focus on the activities germane to the type of software being developed. The SDP-B will also have to address and denote the class of test cases to be used in evaluating the system.

The Software Top Level Design Document (STLDD) and Software Detailed Design Document (SDDD) of the 2167A (draft) are replaced by the preliminary FDD and the updated FDD respectively. The FDDs contain detailed information on the knowledge bases of the system and the inference mechanism used. Along with this detailed information is the inclusion of the reasons for selecting the knowledge representation and reasoning method being used.

The FPS has a similiar format to the Software Product Specification (SPS) of 2167A (draft). Again, additional information to be included in this document relates to knowledge base descriptions. Namely, the Functional Design, Knowledge Base listings, Rule Relationships, KBS Shell listings, Inference Mechanism listing, and any other related listings.

The information contained in the three Software Test Description documents of the 2167A (draft) is included in the KBS STD document. Specific information on how dynamic and static evaluation of the KBS will be performed, is also contained in this document. The STRs of the KBS model follow the same format as the STRs of the 2167A (draft).

3.4.3 Reviews

Table 3.4.3-3 depicts the correlation between 2167/2167A (draft) and KBS reviews. The System Requirements Review (SRR) is not listed in the table as, in hybrid systems, this review should address KBS segment requirements where appropriate.

The System Design Review and Software Specification Review of the 2167/2167A (draft) model are combined into the SD&DR of the KBS model. The items to be reviewed during the SD&DR are listed below :

- KSS
  - System Partitioning
  - Current Set of Requirements

Table 3.4.3-3: Mapping of Reviews and Audits

2167/2167A (Draft)	KBS
System Design Review Software Specification Review	System Design & Demonstration Review
Preliminary Design Review	Preliminary Design & Demonstration Review
Critical Design Review	Critical Design & Demonstration Review(s)
Test Readiness Review	Test Readiness Review
Functional Configuration Audit	Functional Configuration Audit
Physical Configuration Audit	Physical Configuration Audit

3.4.3 Reviews

- IRS
- Prototype Demonstration
  - Domain
  - Knowledge Base
  - Inference Mechanism
  - Knowledge Representation

In reviewing the domain, the focus is on determining if the domain is the appropiate size. The decision being to either enlarge, reduce or retain the domain. The item of knowledge base is addressed to determine if what is there already is correct and what knowledge is missing. The inference mechanism and knowledge representation are reviewed to ascertain the suitability of the methods being used.

The PD&DR of the KBS model is similiar to the Preliminary Design Review of the 2167A (draft) model. The PD&DR should review the following KBS items:

- Preliminary FDD
- SDP-B
- Final Prototype Demonstration
  - Domain
  - Knowledge Base
  - Inference Mechanism
  - Knowledge Representation
- Final Set of Requirements

The prototype issues are reviewed in the same context as during the SD&DR.

The CD&DR of the KBS model is similiar to the Critical Design Review of the 2167A (draft) model. The following are the items to be reviewed during the CD&DR:

- Updated FDD
- Demonstration and Current Status of Implemented System
  - Knowledge Base Coverage
  - Inference Mechanism
  - Knowledge Representation

The review of the knowledge base, representation and inference mechanism items is focused on an as-implemented basis.

The TRR, FCA, and PCA of the KBS model are all essentially the same as their counterparts in the DOD-STD-2167/2167A (draft) models. In hybrid systems the KBS reviews and audits can either be included into the corresponding 2167/2167A (draft) reviews and audits, or they can be performed separately.

Table 3.4.4-4: Mapping of Baselines/Configuration

2167/2167A (Draft)	KBS
Functional Baseline Allocated Baseline	Functional/Allocated Baseline
Developmental Configuration	Developmental Configuration
Product Baseline	Product Baseline

3.4.4 Baselines

Table 3.4.4-4 depicts the correlation between 2167/2167A (draft) and KBS baselines and configuration. As indicated, the Functional and Allocated Baselines of the 2167/2167A (draft) model are combined into one baseline named the Functional/Allocated Baseline. The following products are entered into the noted KBS baselines/configuration:

- Functional/Allocated
  - IRS
  - KSS
- Developmental Configuration
  - FDD
  - Source and Object Code
- Product
  - FPS
  - Updated Source and Object Code

SECTION 4

Recommended Studies/Activities

4.1 Model Application Case Studies

The KBS Developer/Customer Interface model shown in Figure 3–4 was derived from case study data and coupled with DOD acquisition agency needs. Given that an interface mechanism has been developed, the next logical step would be verification of the model. Application of the KBS model on a *live* project would allow one to track activities and identify deviations between the interface model and the actual effort. Once identified, an evaluation could be made to determine whether the deviant activities are unique to the particular project or a deficiency of the KBS interface model.

Application of the model would also enable one to assess the suitability of the recommended products, reviews and baselines. Following the verification effort, the interface model could be adjusted as necessary to better reflect the acquisition/development process.

4.2 Technology Studies

This subsection identifies KBS development areas requiring further study as they pertain to critical system functions and risk reduction efforts. None of these areas were within the scope of work for the current contract.

4.2.1 Critical System Functions

The development of real-time KBS systems needs to be explored further. If KBS tasks cannot be accomplished within the time limits required by a particular system, alternate architectures to remove the KBS tasks from the critical path will have to be investigated.

Decision and control issues need to be defined to highlight areas where KBS's will enhance the capability of the user, as well as functional areas that can be trusted to computer automation. Answers from questions generated by these issues should address how the interface with the man in the loop will be handled, along with the amount of functional autonomy to be programmed into the operational computer system.

Integration of KBS software and the specific hardware implementations designed to support KBS systems is an area needing more scrutiny. The hardware architectures with the greatest near term payoff for the support of KBS functions need to be evaluated in terms of maturity, reliability, functional performance and program release.

4.2.2 Risk Reduction Efforts

Formal testing of KBS software requires further study, especially in the area of acceptance criteria for these nondeterministic systems. Since one of the goals of KBS systems is to work under uncertain conditions, robustness is another level of test that should be addressed in more detail.

Questions about system and software reliability need to be addressed. This will be complicated by the presence of indeterminancy in the KBS software and the type of hardware that may be required to support the KBS systems. There is a need for the study of the mutual effects of software fault tolerance and KBS technology on each other.

Maintenance of KBS will have support requirements and procedures that are different from those currently used in conventional software. An understanding of required maintenance actions must be defined. Tradeoffs in the system design and implementation need to reflect the expected workload in maintaining these programs.

From a management perspective, estimation procedures need to be developed to predict the expected cost of proposed KBS software in terms of effort and dollars. The relevant influential parameters need to be defined in order to produce cost and schedule models which closely represent the development process.

Lastly, the applicability of prototyping techniques to large conventional software systems should be investigated. Given the extensive set of requirements associated with large military systems, prototyping in the Pre-Software Development phase may facilitate the requirements analysis task and reduce the risk of building the wrong or an incomplete system.

4.3 Engineering Discipline

Last but not least, is a need to address areas such as standard languages, support environments and interfaces. For example, is it cost effective to move to Common LISP as a standard language? If so, how is it controlled? Is there reason to believe a standardized support environment consisting of both hardware and software tools will make DOD's job any easier? What interface standards should be promulgated from conventional software to KBS software or vice versa? Is there a need to define data flow and control standards? What about language interface issues?

All the above items are potential candidates for investigation. Each represents a further level of detail below the modeling process which needs to be examined for systems of the future.

Bibliography

[1] Adam, John A. and Fischetti, Mark A. "Star Wars- SDI: the grand experiment". *IEEE Spectrum*, Vol. 22, No. 9, pp. 34–64, September 1985.

[2] Beregi, W. E. "Architecture prototyping in the software engineering environment". *IBM Systems Journal*, vol. 23, No. 1, pp. 4–18, 1984.

[3] Boehm, Barry W., Gray, Terence E., and Seewaldt, Thomas. "Prototyping Versus Specifying: A Multiproject Experiment". *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 3, pp. 290–302, May 1984.

[4] Church, V.E., Card, D.N., Agresti, W.W., and Jordan, Q.L. "An Approach for Assessing Software Prototypes". *ACM SIGSOFT Software Engineering Notes*, Vol. 11, No. 3, pp. 65–76, July 1986.

[5] Clapp, J. A., Hockett, S. M., Preile, M. J., Tallant, A. M., and Triant, D. D. "Expert Systems for C^3I". Mitre Software Center, Contract No. ESD-TR-85-125 Vol. 1, October 1985.

[6] Connell, John and Brice, Linda. "Rapid Prototyping". *Datamation*, Vol. 30, No. 13, pp. 93–100, August 1984.

[7] Correll, John T. "Machines That Think". *Air Force Magazine*, pp. 70–75, July 1986.

[8] Crowley, John D. "The Application Development Process: What's Wrong With It?". *Performance Evaluation Review*, Vol. 10, No. 1, pp. 179–187, Spring 1981.

[9] Daley, Philip C. "C^3I Rapid Prototype Investigation". Martin Marietta Denver Aerospace, Contract No. RADC-TR-85-216, January 1986.

[10] De Marco, Tom. "*Structured Analysis and System Specification*". Yourdon Press, New York, NY, 1978.

[11] Drogin, Edwin M. "AI Issues for Real-Time Systems". *Defense Electronics*, pp. 150–160, June 1986.

[12] Druffel, L.E. and Kernan, Joseph E. and Paige, K.K. and Riski, William A. "Report on the DoD Task Force on Software Problems". Third Draft, July 15, 1982.

[13] Eastport Study Group. "*A Report to the Director, SDIO*". Technical Report, 1985.

[14] Faden, Michael. "Software prototyping - a Misnomer and a Muddle". *Datalink*, pp. 6-9, September 1984.

[15] Gates, K.H., Adelman, L. and Lemmer, J.F. "Management of AI System Software Development for Military Decision Aids". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[16] Glass, Robert L. "Some Thoughts on Prototyping". *System Development*, Vol. 5, No. 8, pp. 7–8, October 1985.

[17] Harmon, Paul and King, David. *"Expert Systems, Artifical Intelligence in Business"*. John Wiley & Sons, New York, NY, 1985.

[18] Honeywell Computer Sciences Center. "RaPIER(Rapid Prototyping to Investigate End-user Requirements)". Contract No. N00014-85-c-0666, March 28, 1986, pp. 1-74, 235-237, 270-274.

[19] Jacob, Robert J. K., and Froscher, Judith N. "Developing a Software Engineering Methodology for Knowledge-Based Systems". Naval Research Laboratory Report 9019, December 17, 1986.

[20] Kauber, Peter G. "Prototyping: Not a Method but a Philosophy". *Journal of Systems Management*, vol. 36, No. 9, pp. 28-33, September 1985.

[21] Lieblein, Edward. "The Department of Defense Software Initiative-A Status Report". *Communications of the ACM*, Vol. 29, No. 8, pp. 734-744, August 1986.

[22] Liebowitz, Jay. "Evaluation of Expert Systems: An Approach and Case Study". In *The Second Conference on Artificial Intelligence Applications , Sponsored by IEEE Computer Society*, December 1985.

[23] Lin, Herbert. "The Development of Software for Ballistic-Missile Defense. *Scientific American*, Vol. 253, No. 6, pp. 46-53, December 1985.

[24] Lockheed-Georgia Company. "Software Development Plan for the Pilot's Associate Porgram". Contract No. F33615-85-C-3804, July 7, 1986.

[25] Lynch, Frank, Marshall, Charles, O'Connor, Dennis and Kiskiel II, Mike. "AI in Manufacturing at Digital". *The AI Magazine*, Vol. 7, No. 5, pp. 53-57, Winter 1986.

[26] Matsumoto, Yoshihiro. "Management of Industrial Software Production". *Computer*, Vol. 17, No. 2, pp. 59-70, February 1984.

[27] McGraw, Karen L. and Bruce A. "The Phantom Crew - AI in the Cockpit". *DS and E*, pp. 44-54, November 1985.

[28] Metzger, P.W. *"Managing a Programming Project"*. Prentice-Hall, Englewood Cliffs, N.J., 1973.

[29] Peters, Lawrence J. *"Software Design: Methods & Techniques"*. Yourdon Press, New York, NY, 1981.

[30] Powell, Christopher A., Pickering, Cynthia K. and Keith T. Wescourt. "System Integration of Knowledge-Based Maintenance Aids. In *Fifth National Conference on Artificial Intelligence, Sponsored by AAAI*, December 1986.

[31] Pressman, Roger S. *"Software Engineering: A Practitioner's Approach"*. McGraw Hill Book Company, United States, 1982.

[32] Sanders Associates Inc. "Software/Microprocessor Task Force". Final Report, May 6, 1986, pp. 33-36.

[33] Sarvari, I. L. "The Case For PROTOTYPING In Systems Development". *Canadian Datasystems*, Vol. 15, No. 10, pp. 100-104, October 1983.

[34] Schach, Stephen R. "Prototyping, Design and Early Cost Estimation Problem". In *Third International Workshop on Software Specification and Design, Sponsored by IEEE Computer Society*, August 1985.

[35] Segall, Mark J. "The Use of Prototyping to Aid Implementation of an On-Line System". *Systems, Objectives, Solution*, Vol. 4, No. 3, pp. 141–156, August 1984.

[36] Somin, Herbert A. "Whether Software Engineering Needs to be Artificially Intelligent". *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, pp. 726–732, July 1986.

[37] Taylor, Tamara and Standish, Thomas A. "Initial Thoughts on Rapid Prototyping Techniques". *ACM SIGSOFT Software Engineering Notes*, Vol. 7, No. 5, pp. 160–166, December 1982.

[38] Weiser, Mark. "Scale Models and Rapid Prototyping". *ACM SIGSOFT Software Engineering Notes*, Vol. 7, No. 5, pp. 181–185, December 1982.

[39] Wheildon, David. "Prototyping: Shortcut To Applications". *Computer Decisions*, vol. 16, No. 7, pp. 138–142, 146–147, June 1984.

[40] Yourdon, Edward. "*Techniques of Program Structure and Design*". Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

Acronyms

CDR	Critical Design Review
CD&DR	Critical Design and Demonstration Review
CPCI	Computer Program Configuration Item
CRISD	Computer Resources Integrated Support Document
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSDM	Computer System Diagnostic Manual
CSOM	Computer System Operator's Manual
DBDD	Data Base Design Document
DID	Data Item Description
DOD	Department of Defense
ECP	Engineering Change Proposal
FCA	Functional Configuration Audit
FDD	Functional Design Document
FPS	Functional Product Specification
FSM	Firmware Support Manual
HIPO	Hierarchical Input-Processing-Output
HWCI	Hardware Configuration Item
IDD	Interface Design Document
IRS	Interface Requirements Specification
JLC	Joint Logistics Commanders
KA	Knowledge Acquisition
KBS	Knowledge Based Systems
KR	Knowledge Representation
KSS	KBS Segment Specification
OCD	Operational Concept Document
OTS	Off-the-shelf
PCA	Physical Configuration Audit
PDL	Program Design Language
PDR	Preliminary Design Review
PD&DR	Preliminary Design and Demonstration Review
PIRS	Preliminary Interface Requirements Specification
PKSS	Preliminary KBS Segment Specification
PSRS	Preliminary Software Requirements Specification
SDDD	Software Detailed Design Document
SDF	Software Development Folder
SDP	Software Development Plan
SDP-A	Software Development Plan A
SDP-B	Software Development Plan B
SDR	System Design Review
SD&DR	System Design and Demonstration Review

Acronyms

SPM	Software Programmers Manual
SPS	Software Product Specification
SQE	Software Quality Evaluation
SQEP	Software Quality Evaluation Plan
SRR	System Requirements Review
SRS	Software Requirements Specification
SSR	Software Specification Review
SSS	System Segment Specification
STD	Software Test Description/Procedures
STLDD	Software Top Level Design Document
STP	Software Test Plan
STPR	Software Test Procedure
STR	Software Test Report
SUM	Software User's Manual
TRR	Test Readiness Review

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END
DATED
FILM
8—88
DTIC